

Put It Together

Animating Machine Assembly Instructions for Novices

by

Pok Yin LEUNG

Bachelor of Arts in Architectural Studies BA(AS)
The University of Hong Kong ,2011

Submitted to the
Department of Architecture in partial fulfillment
of the requirements for the degree of

**Master of Science in
Architecture Studies**

at the
**Massachusetts
Institute of
Technology**

June 2016

©2016 Pok Yin Leung. All rights reserved.

The author hereby grants to MIT permission
to reproduce and to distribute publicly paper
and electronic copies of this thesis document in
whole or in part in any medium now known or
hereafter created.

Signature of Author:

Victor Leung

Department of Architecture
May 18, 2016

Certified by:

Terry Knight

Professor of Design and Computation
Thesis Advisor

Accepted by:

Takehiko Nagakura

Associate Professor of Design and Computation
Chair of the Department Committee on Graduate
Students

I am indebted to Terry Knight for her help and advice throughout the course of this research. Her thoughtful comments and continuous support had been most helpful during my years at MIT. Her speed of reading text still amazes me.

Thanks to Alexander Slocum and Neil Gershenfeld for their insightful contextualization of my research. Their opinion was very helpful for me to see the bigger picture.

Thanks to Takehiko Nagakura for his patient advice and the opportunity to work with him. He treated me as a friend more than as a student. I wish I hadn't crash the drone in Italy.

Thanks to Diego Pinochet who has been an inspiring friend, providing me with joyful conversations. His polished presentation graphics and video influenced me, both in good and bad ways. I wish I had spent more time with him.

Thanks to Inés Ariza and the relaxing coffee breaks we had. It has been a very joyful time with her during the last months of this journey.

Thanks to Alexandros Charidis who keep saying there are methods online for everything. He wasn't very patient but still explained difficult mathematics and design space concepts to me.

Thanks to many other supporting staff and technical instructors who offered me kindness and help; their actions may have been simple but they meant a lot: Justin A. Lavalley from the workshop, Marilyn Levine from the writing center, Cynthia Stewart, our administrator, staff from the MIT-SUTD Collaboration and staff from Makeblock.

Finally, I wish to thank my parents Susanna Yu and Romeo Leung, who have always given me their love and support. I hope I made you proud.

Thesis Committee

Terry Knight

Professor of Design and Computation
Thesis Advisor

Neil Gershenfeld

Professor of Media Arts and Sciences
Thesis Reader

Alexander Slocum

Professor of Mechanical Engineering
Thesis Reader

Table of Content

Introduction	7
Technical Background	13
Put It Together - Assembly Instructions Animated	21
CNC Workshops for Novices - Learning Experiment	47
Observations and Interpretations	55
Content preparation guidelines	73
Conclusion	79
Future Work	83
Appendix	89
Reference	99

Put It Together

Animating Machine Assembly Instructions for Novices

by **Leung Pok Yin**

Submitted to the Department of Architecture on May 18, 2016 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Architecture

Abstract

We are no longer satisfied with rapid prototyping machines! The new frontier in digital fabrication is the rapid prototyping of rapid prototyping machines. Using modular electronics and robotic parts, the essence of machine making now lies in part assembly. With the advancement of online education, how will schools teach part assembly? How will Makers share the knowledge of putting things together?

Traditional assembly instructions designed with text, diagrams and images are often not effective in showing complex assembly motions, and are poorly adapted to large complex machines. Demonstration videos are expensive to produce, and they are limited to a single camera view.

Put It Together is a new digital workflow that consists of two parts: (1) A CAD plugin that allows machine designers to easily create assembly animations, and (2) an interactive web player that allows novices to view the animation. Starting with a CAD model, designers can easily create and edit an animation using a visual graph. The software

interprets the graph and creates a step-by-step 3D animation. Novices can view the animation using a web browser, interact with the viewing angle, and progress at their own pace.

The web player was tested, developed, and evaluated through multiple workshops in which students learned machine assembly with successive versions of the player, and proved its value in an educational environment. Other potential applications of the Put It Together approach, beyond machine assembly, include self-assembled furniture, DIY projects and toys.

Thesis Supervisor: Terry Knight

Title: Professor of Design and Computation

1.1 About this thesis

1.2 Mass-customization and fabrication machines

1.3 Relevance to DIY culture and maker's movement

1.4 Relevance to design education

1.5 Open education

1.6 Assembly instructions

1.7 Contribution of this thesis

1

Introduction

Today, we have significant research about personal fabrication. Laser cutters and 3D printers are getting cheaper and easier to use and they are slowly getting into the homes of people as a desktop fabrication machine. However, existing fabrication machines only cover a small palette of processes: 3D printers extrude, laser cutter fire laser, and water jet cutter shoots water.... There are many other manufacturing and industrial processes that are not covered by these general purpose machines.

A more powerful approach towards digital fabrication, enabling people to build anything, is to enable them to build the very machine that makes things. Just as a good carpenter makes his own tools, a good designer should also be interested in the processes and machines that make their products.

I believe modular hardware¹, modular electronics² and open source software³ will lower the barrier of making machines and shift the focus of machine making towards assembly. However, CNC machines have many parts, and assembling them requires many tedious steps. **The problem is that current methods that show instructions are either difficult for machine designers to produce, or difficult for readers to understand.** An effective method for designing instructions is needed to communicate how to assemble these complex machines.

I started teaching workshops in early 2015 using a robotic hardware made by Makeblock, electronics using Arduino, and using open source controlling software grbl. These workshops aim to teach designers and makers — with no engineering background — how to make CNC machines that make things. Because those classes were rather large, I gave students a step by step illustrated hand-out to explain the assembly process, so students could assemble it at their own pace. But I quickly realized that students were making all sorts of mistakes because the instructions are causing a lot of confusion.

1 Products such as Makeblock construction system, Vex Robotics and Lego Mindstorm

2 Open source projects and products such as Arduino microcontrollers and Raspberry Pi are cheap controllers available for quick customization. Gestalt framework (Moyer 2013) integrates software and hardware control into modularized network nodes for quicker and easier development of new fabrication machine control.

3 Open source control software such as grbl and tinyg2 are easy to use, multipurpose G-Code controllers.

1.1 About this thesis

With this thesis, I propose a new method to replace paper instructions using an Interactive 3D animation. This solves a fundamental limitation associated to paper instructions, where viewers can only see snapshots of the assembly process, and also only from a fixed angle. With today's WebGL and JavaScript technology, we are able to create a step by step animation where assembly movements no longer require drawing arrows, and viewers can advance the animation according to their own learning speed. They are also able to interact with the viewing angle and see all sides of the 3D geometry.

Machine designers can easily convert their existing CAD assembly models into the instruction animation by using a special CAD plugin. Through a graphical interface, designers create an assembly graph that represents connective relationships and assembly actions between parts. The plugin interprets the graph and creates the instruction animation automatically.

1.2 Mass-customization and fabrication machines

Traditional manufacturing design optimizes one product that fits the widest audience. This “one size fits all” model has been surpassed by recent applications of CNC machines in manufacturing, which allow products to be mass customized at a reasonable cost. The current Fab Lab movement aims to standardize CAD/CAM software and fabrication tools to create a workspace that aims to make almost anything. However, this updated

model is still limited by what those machines can make. The next step towards customized production and design freedom depends on developing novel machines.

Recent developments in modular robotic parts (such as Makeblock and VEX) and open source software and electronics (such as grbl and tinyg2 which run on Arduino Uno and Due) have offered an opportunity for rapidly designing and making CNC machines. These off-the-shelf parts obviate much of the part-making process, allowing machine-making to focus on an effective combination of these parts.

1.3 Relevance to DIY culture and maker's movement

This culture of “rapid prototyping of rapid prototyping machines”⁴ was quickly embraced by the Maker's Movement whereby designers and hobbyists with no background in engineering are motivated by a desire to make their customized CNC machines or cheaper-than-market CNC machines. This entails an unconventional method of learning how to design and make machines.

In the context of ready-designed machines such as DIY kits and open source machine designs⁵, the engineer often prepares assembly instructions in the form of texts, images or videos for novices to follow. In the context of custom-designed ma-

chines, hobby designers often rely on web blogs and web forums to exchange experience and information. In both of these scenarios, the hands-on experience in assembling machines is hard to communicate. The trendy Maker Faire events with their slogan “The Greatest Show (& Tell) on Earth”, perhaps address some of this problem by encouraging makers to gather and interact with the people and their creations in person. However, these events only accommodate the privileged few who can travel to the venue.

1.4 Relevance to design education

It is common for students to study and visit good buildings as part of their education in architectural design. Case-based reasoning theory describes such a learning model, where the knowledge and intuition of solving new problems can be based on the solutions of similar past problems.

In order to study complex objects with many parts, such as machine design, product design, furniture design, and architecture, it is beneficial to study the spatial relationships and the assembly process of these objects. We have seen many examples of the use of exploded diagrams, annotated illustrations and cut-away drawings in engineering and architecture textbooks. One good example is David Macaulay's series of illustrated books in 1970s (Cathedral: The Story of Its Construction 1973) that explain how things work. His careful selection of drawing styles, viewing angles and cut away positions made technical and non-technical readers enjoy it alike.

4 A phrase coined by Neil Gershenfeld to describe the impact of Fab Lab.

5 RepRep 3D printer is an example of an open source machine that aims to be easily made by novices.

1.5 Open education

Motivated by the same belief of Open Education, this thesis aims to develop methods of sharing knowledge and experience that are accessible and understandable by most people. The recent trend in developing more types of course content for Massive Open Online Course (MOOC) prompts us to rethink how practical lab demonstrations and hands-on experience can be recorded and communicated via the browser. And in such a process, exploring new opportunities are offered by this new medium.

Instructables.com provides an online platform for knowledge exchange between designers and novices. The platform allows designers to upload text and images to explain how to do or make things. Since its start in 2005, it contains more than 200,000 user submitted tutorials and receives 4 million visitors a month. Most of the tutorials involve some form of assembly. **There clearly isn't a lack of hobbyists willing to share instructions. What is lacking is a good method, other than just text and images, to communicate how to assemble.**

1.6 Assembly instructions

Recently some designers who are capable of creating CAD models embraced the open-source or free license movement. They share their work online in repositories such as GrabCAD and 3D Warehouse. While these CAD models are effective at communicating shapes and spatial relationships, common CAD software (such as Inventor, SketchUp, Rhino, TinkerCAD) and CAD standards used

by novice makers do not emphasize how parts are assembled together. On the other hand, professional CAD systems are expensive and difficult to learn, and they also assume their users to be engineers and that the assembly logic can be inferred from static 3D models. This is often not true for novice makers.

This topic of assembly instructions that I'm trying to improve, is applicable not only to machines. It can also be used in self-assembled products such as DIY projects, flat-pack furniture and user-assembled architecture such as shelters and DIY housing. **It is a means of communicating "how to assemble"**. It is also a valuable tool in design education where it can be used to show parts being put together in the most illustrative way. In the case where taking apart real objects is not possible, an interactive animation is a cost effective way to study many examples.

In the context of ready-designed machines and objects, a one-to-many relationship exists between a corporation and their customers. These companies are able to afford the effort to produce high quality texts, drawings, images and videos to convey the assembly instruction of their products. For example, IKEA produces paper-based instructions for their self-assembled furniture. They are famous for their non-language specific instructions and graphic clarity. Lego produces colourful instruction books for kids to follow and construct pre-designed objects. However, even with careful design, it is not uncommon to hear users complaining about confusing instructions.

In the context of individual hobbyists sharing their designs, it is often too much effort for them

to bother making illustrations and video. This leads to an overwhelming focus in their blog and forum posts to show off superficial images of their projects without much useful information that other hobbyists can learn from.

1.7 Contribution of this thesis

This thesis proposes an alternative medium to communicate assembly instructions using web animation. This replaces or augments the traditional use of texts, photos and videos in communicating movements during the assembly process.

There are three main contributions. (1) **Tools for designers (authors of the instructions) to create and edit the animation.** I designed and developed an assembly relationship graph and the associated algorithms for designers to create and edit the relationships between parts. These relationships are then translated into animation movements; (2) **A file format to encode and share the animation online.** I developed a schema to encode this animation. (3) **A software for novices to playback and interact with the animation.** I developed a Web Player that is easy to learn and easy to use for novices, and I tested it in an educational environment.

This thesis targets two audiences at the same time, ensuring the workflow is meaningful to both parties of the cycle: (1) Designers who design machines (or any other products) and their assembly instructions. (2) Novices who follow instructions to assemble machines (or any other products).

To validate this new instruction method, I taught a number of hands-on workshops to observe students' reactions to this new medium. This thesis provides a written record of the problems students encountered, and offers insights into their causes and solutions. These observations will help direct future software implementations and provide guidelines for creating contents.

I named the whole software package *Put It Together* to reflect the ease and joy of both designers and novices using this software.

2.1 Assembly (Computer science and robotics)

2.2 Part models and libraries

2.3 Computer animation

2.4 CAD software and animation

2.5 Camera

2.6 Online CAD viewer

2.7 Three.js

2.8 Rendering colour and materials and styles

2.9 Cognitive psychology

2

Technical Background

This section gives a brief background of the technical issues related to this thesis. A brief explanation is included about the similarity and difference to the method I propose.

2.1 Assembly (Computer science and robotics)

Assembly is a classic problem in robotics. Researchers in computer science and robotics have conducted research on automatic assembly sequence planning and automatic instruction presentation. Although the scope of this thesis does not include sequence planning, they share similar issues.

Maneesh Agrawala et al (2003) have developed algorithms to produce assembly instructions for any given object geometry, orientation, and

optional grouping and ordering constraints on the assembly's parts. Their system automatically finds and optimizes the assembly sequence and camera angle based on cognitive psychology for effective visual communication. It can produce assembly instructions as graphical illustrations. My proposed tool differs by presenting the information in an interactive 3D animation in a browser based environment thus reducing the need to search for the best viewing angle. Yet, his quantifiable cognitive approach is useful when applied to other problems.

Schulz et al (2004) used graphs for their "Data-driven fabrication design software" to keep

track of the relationships between parts. My system implements a similar graph but with the fundamental difference that I do not hide this graph internal to the software. **Authors can see this graph directly to understand the order of assembly actions and they are able to interact with this graph to add, remove and change the sequence of the operations.** Schulz also separated the “connecting parts” and the “principal parts” to simplify the design process. In his definition, connecting parts (such as screws and brackets) are automatically inferred from the placement of the “principle parts”. My proposed software does not make this clear cut separation. Because in practical design applications, it is possible to use the two interchangeably (e.g. when a screw is designed to act as a mechanical stop, it is not connecting two parts together but it is a part of itself). Instead, I represent these fixture parts as hidden “part” nodes that are linked by the “action” node.

2.2 Part models and libraries

CAD models for mechanical parts are increasingly available for free, on online platforms such as GrabCAD, Thingiverse and Google 3D warehouse. This phenomenon is driven by two separate factors. The first comes from the parts manufacturers and vendors: they are motivated to provide CAD models of their products, free of charge, to make it easy for engineers to design machines with their products in their CAD design workflow. For example McMaster-Carr has over 100,000 CAD models freely available from their online catalogue.

Another motivation comes from the show and tell culture of the Maker’s movement. This culture views the act of sharing designs as a contribution to the movement itself. Makers who designed various types of objects share their CAD models with open license.

This movement towards free access of almost any part models assured the premise of an ecosystem where machine design effort lies in the combination of these parts. Machine designers can focus on the combination of these parts and create custom component only when it is necessary.

2.3 Computer animation

3D Computer animation is a long existing field in computer graphics. A generic description of animation is “A change in model attributes over time”. These attributes can be position, rotation, size, colour, opacity and many more. Typically, animators manipulate polygon mesh models and create key frames on a timeline, and then a computer software interpolates the motion and renders all the still frames in between.

Because most mechanical parts are non-deformable, we can simplify the animation problem by interpolating an affine transformation across time. In a 3D modelling environment, an affine transformation is typically represented using a 4 by 4 matrix. However, linearly interpolating the values in the matrix will not produce smooth motion. Erik et al. (1998) provided a comprehensive study for animation interpolation. They described a method of interpolating rotations with quaternions to achieve a visually pleasing trajectory.

Assembling deformable parts such as springs requires deformation of the mesh model, and additive and reductive processes such as gluing, welding and cutting requires changing the model.

The list below shows examples of actions found in mechanical assembly that can be categorized into the four types.

Rigid body transformation:

- Inserting pin, screws, washers, nails into each other (translation)
- Mating two parts together (translation)
- Tightening screws, nuts, set screws. (translation + rotation)
- Inserting shafts and rods through bearings, linear guide rails (translation)
- Inserting nut into lead screw (translation + rotation)
- Taping

Deformation:

- Installing retaining ring, clips, rivets (deformation + translation)
- Bending, wiring and routing cables, belts, chains, tapes (deformation)
- Compressing and preloading springs (deformation)
- Tightening spring washer (deformation)
- Installing zip ties, sticky tape, cable ties (deformation)

Additive Modification:

- Gluing parts with contact glue / super glue
 - Gluing with hot glue / epoxy glue / filler
 - Welding metal
 - Painting

Reductive Modification

- Cutting cables, belts, chains, tapes
- Stripping electrical cables, plastic coated cables
- Carving, milling, turning, sawing and other machining process

In this thesis, I focus on the assembly process that can be animated without deformation.

In the context of machine making by hobbyist, it is common to use modular robotic parts that do not require deformation and machining. It is also common to find parts that can be easily detached for reuse or modification, such as in robotic sets. For example screws are favoured against deformable rivets because they can be easily undone. For that reason, this thesis will focus on affine transformation animation and consider other animation types as future work.

2.4 CAD software and animation

Animating CAD models is not a new idea. It has been implemented in many professional CAD systems such as Solidworks, Inventor and CATIA.

Autodesk Inventor contains an animation feature that deals with presenting robotic kinematic movement. It is designed for animating robotic

joints to demonstrate the operation of a robot. Their method allows the user to have fine control over the animation of individual objects. However, this fine control requires the designers to plan carefully and make a lot of decisions to set up one movement. Thus it is very time consuming to animate the many steps required for machine assembly. My software allows machine designers to pick the relationship between two parts (such as bolting), and the tool automatically infers the animations parameters (position of the various parts and the bolt before and after the action, animation speed, camera position, appearance of the screwdriver and wrench). The software also allows users to edit the assembly sequence through a tree like structure through a graphical user interface and the animation will be automatically modified.

2.5 Camera

3D animation requires a theoretical camera to be placed in a scene with the 3D models. The computer renders each frame of the animation from the perspective of this theoretical camera. There are two common camera types, corresponding to their method of 3D to 2D projection: Orthographic Camera and Perspective Camera. Orthographic projections is a common method for technical drawing because it is easy to draw by hand and the 2D drawing preserves scale and measurability. Perspective projection is more closely related to what the human eye sees. It gives an immersive viewing experience and better perception of depth. In this thesis, I have chosen to use Perspective Camera because most CAD novices are more familiar with it from their exposure to 3D animation and first-person video games. It is also

unlikely for them to take measurement from the screen.

In a multi-step assembly instruction, the location of interest will change and require adjusting the camera position and angle. For traditional paper-based illustration, Agrawala et al. (2003) proposed an algorithm that decides when to change the viewing orientation in generating 2D assembly instructions. The intention is to improve the visibility of subsequent parts or take the natural orientation for the object. My proposed system allows the viewer to interact with the camera with a mouse drag gesture, thus the model can be viewed from any angle. However, it is still beneficial to provide a suggested viewing angle at the beginning of each step; ensuring viewers are aware of all the components that need to be installed.

2.6 Online CAD viewer

GrabCAD and Google 3D Warehouse are online 3D model repositories for open-source models. They feature web-based 3D CAD model viewers to allow readers to interact with the model. Both viewers are programmed with WebGL libraries and Three.js libraries. Google's 3D viewer is based on SketchUp's online viewer. It contains only camera controls for changing camera position. GrabCAD's viewer implementation also allows sectioning and measurement. However none of these platforms allow animation.

2.7 Three.js

The recent trend in internet-browser applications proved the feasibility of using browsers to present 3D interactive animation to a large audience. The Web Player in *Put It Together* takes advantage of recently available WebGL libraries and Three.js framework to create a browser based interface for displaying 3D graphic animation.

Three.js allows developers to setup a scene with complex lighting and 3D models and provide rendering wrappers from OpenGL libraries. Combining JavaScript's ability to gather user input from mouse and keyboard and dynamic html document alteration, it is currently the best tool to build 3D-rich online applications.

2.8 Rendering colour and materials and styles

Visual communication is a matter of cognitive theory. It concerns how people understand drawings and animation.

The developed web interface uses OpenGL rendering features to render frames to create an animation. Various rendering materials and styles offer different stylistic approaches to render the CAD models. WebGL libraries allowed me to test different colouring schemes and materials with little programming effort. I was able to test grey scale materials, real colour materials, transparent materials and outlines.

While photorealistic raytracing rendering and complex lighting techniques allow a realistic depiction of the model, they consume more computing power and result in a low frame rate. Because assembly animation concerns little with realism, I chose to work with simple materials and attempted non-photorealistic styles. This aims to guide the reader's attention towards the assembly instead of the parts.

Transparency, selective rendering style, post rendering effects and outlines are used to direct or focus viewers' attention. For example, Bruckner and Gröller (2007) described a method to render halos around 3D models to enhance depth perception. Further discussion and implementation can be found in the next chapter.

2.9 Cognitive psychology

Mitra et al. (2013) presented several visual techniques to convey the movements of parts in their work on communicating mechanical linkages: (1) motion arrows that are not obscured, (2) careful selection of frame sequences that highlight key snapshots of complex motion and (3) the use of animation. While (1) and (2) are applicable to 2D graphics, they point out the need to evaluate the method's effectiveness with cognitive assessments.

Agrawala et al. (2003) pointed out that "repetitive operations in detail can make the instructions unnecessarily long and tiresome. A better approach is to skip repetitive operations after they have been presented in detail a few times." Their approach is to group parts that require similar

attachment operations as similar-action groups. In this thesis I have tested various methods when I created the animation. The results are summarized in section 6.1.

3.1 Assembly graph

3.2 CAD Plugin - for Rhino 5

3.3 From graph to animation

3.4 Web Player

3

Put It Together - Assembly Instructions Animated

Put It Together is a software package that consists of two parts, each with a different audience — machine designers and novice machine makers. For the machine designers, I propose a CAD Plugin to easily create an assembly animation by specifying the relationships of the assembled parts. Section 3.1 discuss a novel method to represent animation sequence, Section 3.2 describe implementation details for the CAD plugin and Section 3.3 describe an encoding format for the animation for web exchange.

For the novice makers, I propose a Web Player to view the animation from a web browser. In section 3.4, I describe the implementation details for the Web Player.

3.1 Assembly graph

The creation of CAD models is outside the scope of this thesis. This thesis focuses on the relationship between the parts within an assembly. The role of the CAD plugin⁶ is to allow machine designers to define the assembly logic after a CAD model is created. The assembly logic includes **Connectivity relationships** and **sequence relationships**. The CAD plugin will then compute the object visibility, position and movement for each step and generate keyframes for the final anima-

⁶ A plugin is a small software component that functions with a larger software with the intent of adding more functionality.

tion. The plugin will encode the animation in a file format (explained in next chapter) for the web player to playback.

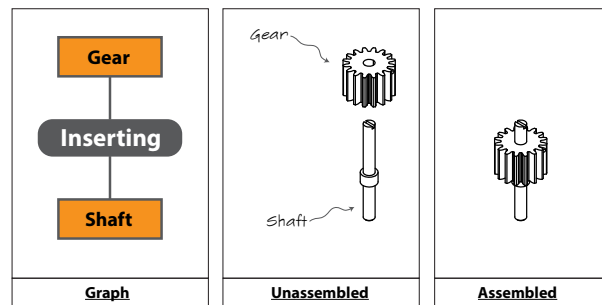
In this thesis, I have implemented this software as a plugin for Rhinoceros 5.0. However, the concepts and algorithms of this CAD plugin can be applied to many other CAD platforms for different types of assemblies. Therefore, I will describe the algorithms in a generalizable way, not specific to a particular CAD software. I will be using machine construction as an example to explain its functions, but the same logic can be applied to many other objects.

Defining parts and actions

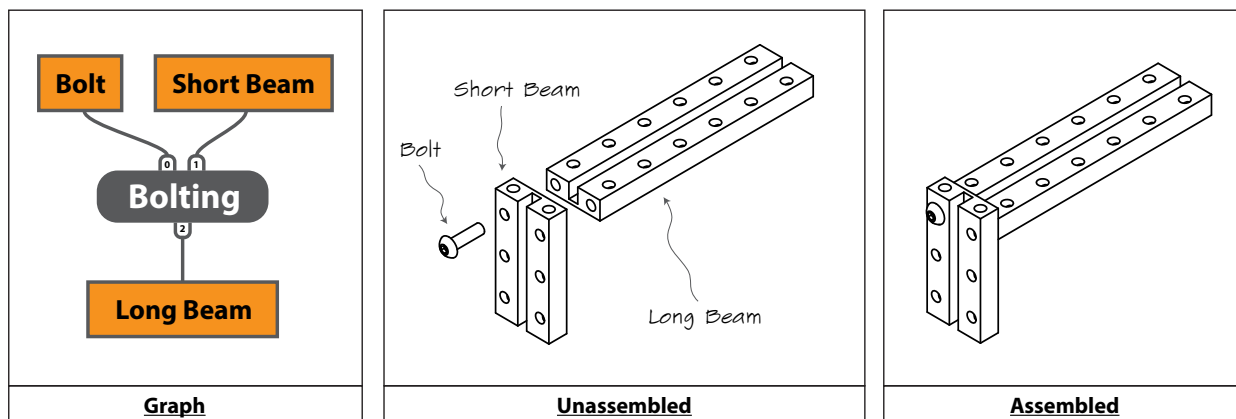
In order to describe the relationships between parts, I propose a graph structure with two types of nodes. **Part nodes** represent physical parts in the assembly, for example a screw, a plate, a bracket or a motor. Each part node should have a one to one relationship with each physical occurrence of a part. For example, in a flat pack furniture, a part represents one loose component

that is delivered to the user, even if that part was pre-assembled with multiple parts.

Action nodes represent the assembly actions that connects two parts together, for example: Bolting, Inserting, Placing and Tightening. Each action represents the operation on one or more Part Nodes, thus edges can be drawn between an Action Node and its related Part Nodes. Below is an example of an Inserting Action between a gear and a shaft.



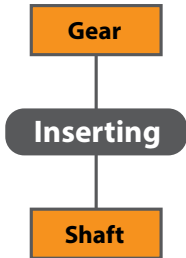
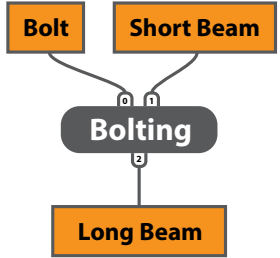
Below is an example of a Bolting Action describing the relationship between a bolt and two beams. Numbering labels are added next to the Action Node to denote the sequence of the bolting stack, which will be explained later.



Each type of action has a specific set of properties to describe the action. For example, the Inserting Action requires a movement vector, the Bolting Action requires the center line of the bolt, and an ordered list of the parts in the bolt stack. This information may be automatically inferred from the CAD model, or can be specified by the user man-

ually⁷ when creating the Action Node, users may later choose to override these properties manually.

⁷ For example, user can pick the Part Nodes by clicking on the CAD models, in the sequence of how the bolt stack is assembled.

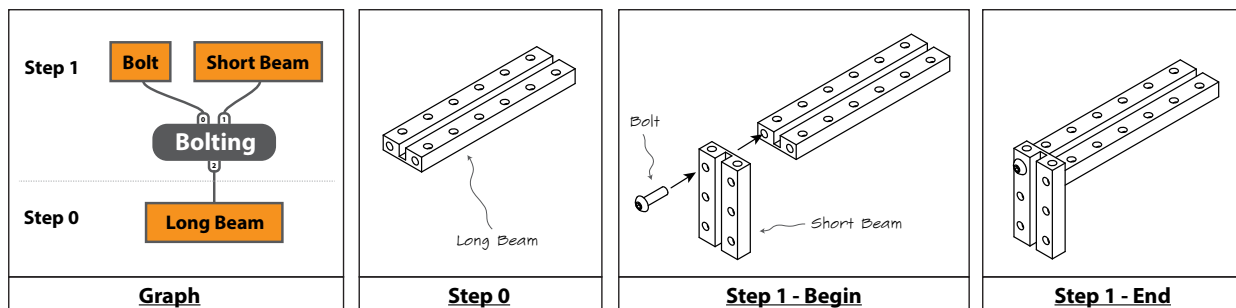
	<p>Inserting Direction:</p> <ul style="list-style-type: none"> - Unit Vector (0,0,-1) - Distance (30mm) <p>Active Node:</p> <ul style="list-style-type: none"> - Gear Node <p>Receiving Node:</p> <ul style="list-style-type: none"> - Shaft Node 		<p>Bolting Direction:</p> <ul style="list-style-type: none"> - Unit Vector (0,1,0) <p>Nodes in Stack:</p> <ul style="list-style-type: none"> - Bolt Node - Small Beam Node - Large Beam Node <p>Node Movement:</p> <ul style="list-style-type: none"> - Distance (50mm) - Distance (20mm) - Distance (0mm)
<p>Inserting Action</p>	<p>Properties of Inserting Node</p>	<p>Bolting Action</p>	<p>Properties of Bolting Node</p>

Defining assembly sequence

The vertical **arrangement** of the nodes describes the assembly sequence. The first step is step 0⁸,

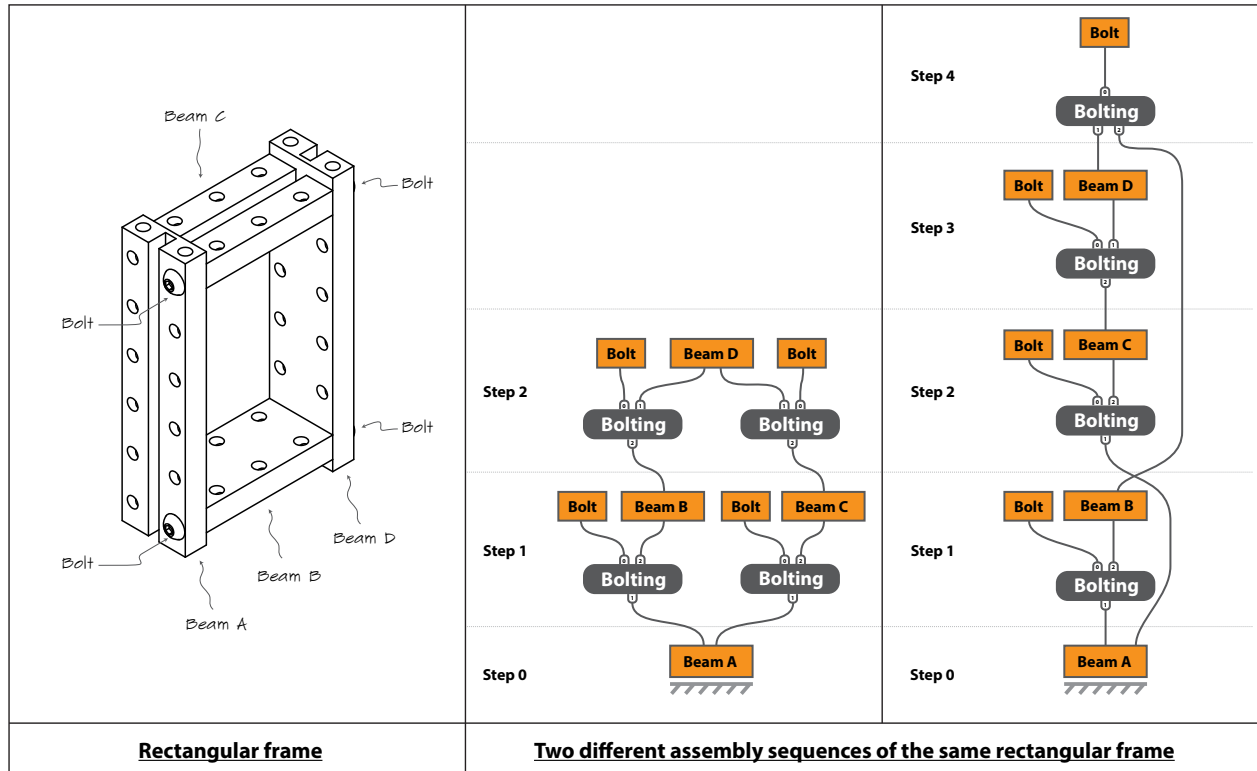
⁸ The first step is named step 0 because no action can be specified on the first step. It is a step that is reserved for the first part(s) for the first action to be placed in the scene.

which starts at the bottom of the graph. Each step contains one or more Action Nodes that represent the actions to be performed in that step. The resulting animation will animate the parts according to the information defined in each step. The example below is a bolting action, represented using a traditional paper-based illustration technique with arrows indicating the direction of movement.



It is common to have connections from a Part Node to multiple Action Nodes. For example, a rectangular frame assembled from 4 Part Nodes and 4 actions have connections like a loop. The arrangement of the Nodes in the vertical order determines the sequence in which the parts are assembled and how the assembly will be ani-

mated. It is up to the designer to decide how to arrange this sequence based on the designer's experience, although some guidelines are proposed later in this thesis which can offer suggestions to the designer from the software. Below is an example of a rectangular frame, assembled in two different ways.



Note: The connections in the two graph are identical, only the arrangements are different. Node to node connection represent actual relationships between parts while vertical arrangements represent assembly sequence. Horizontal arrangements is only for aesthetical layout.

Defining tools

It is possible to store and provide information about the tools that are required for each action in the assembly. For example, Bolting Action al-

most always requires a screwdriver. If a nut is present, then a wrench is also needed. Because tool information is closely associated to the part itself (for example, an M3 screw and M4 screw require different drivers), it is possible to store the tool information as a part property in the Part Library⁹, which is provided by manufacturers. This includes one or more pointers to instances in a Tool Library and the tool position relative to the part.

⁹ Part library will be introduced in next the few pages.

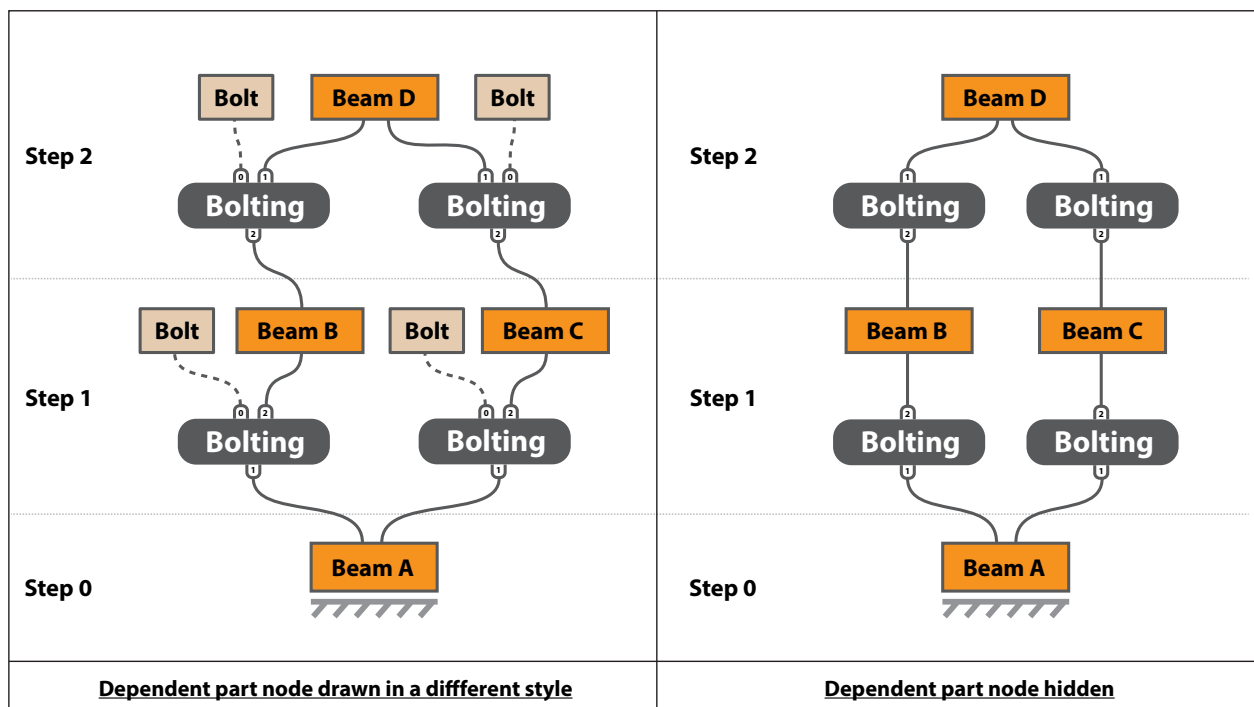
This tool information allows the Web Player to display a list of tool names or images during the animation playback, or animate the tool together with the animation.

In this thesis, using machine assembly as examples, screwdrivers are not animated. This is because if tools are animated together with the parts, each part in each step has to be animated separately, otherwise there will be multiple instances of the same tool shown at the same time and these might also clash with each other. This would produce a rather lengthy, complex animation. However, the possibility of animating tools remains open for future development where applications demand.

Simplifying the graph

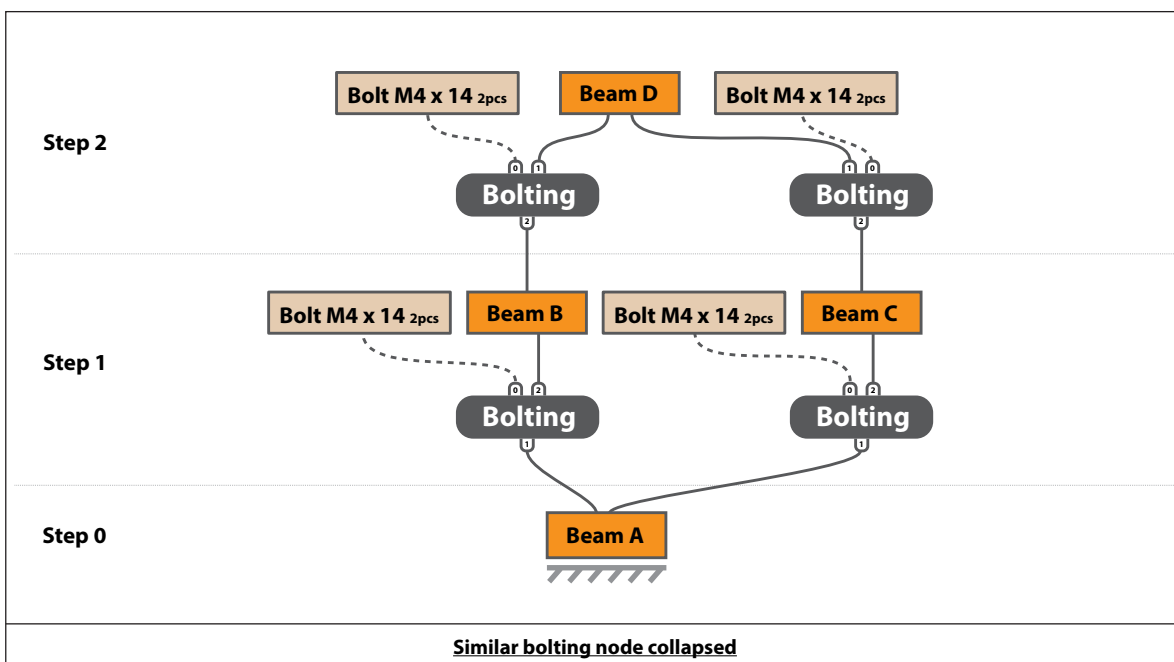
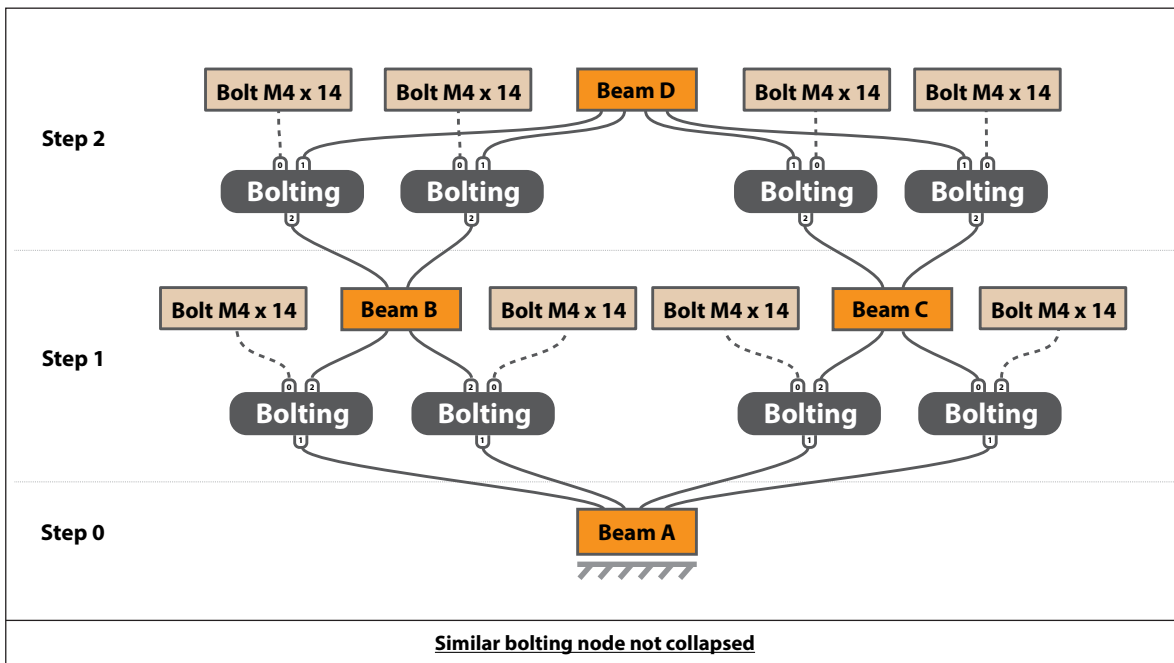
In a machine assembly, it is common to have many fasteners such as screws, washers and nuts. These fasteners always have a one to one relationship with a Bolting Action and they do not have connectivity with any other parts. Because it is impossible for these Part Nodes to make more than one connection and it is rare for the design-

er to edit them independently, it is possible to distinguish these parts with a different UI style, and simplify the visual appearance of the assembly graph by hiding these components. The graph below is the same graph as the previous example, but “Dependent Part Nodes”, the bolts, are drawn in a different style and another graph where they are hidden. Note the hidden version is clearer for the designer to edit.



For the same reason, we can also combine similar screws that have identical bolting parameters into one Bolting Action and one Part Node. This allows similar bolts to share parameters and simplify the graph editing process. The following graph shows a rectangular frame similar to the previous exam-

ple. Instead of one screw per connection, there are two screws per connection. The graph on the left have one Bolting Action Node per bolt and the graph on the right have combined two similar Action Nodes into one representation.



Detecting an invalid graph

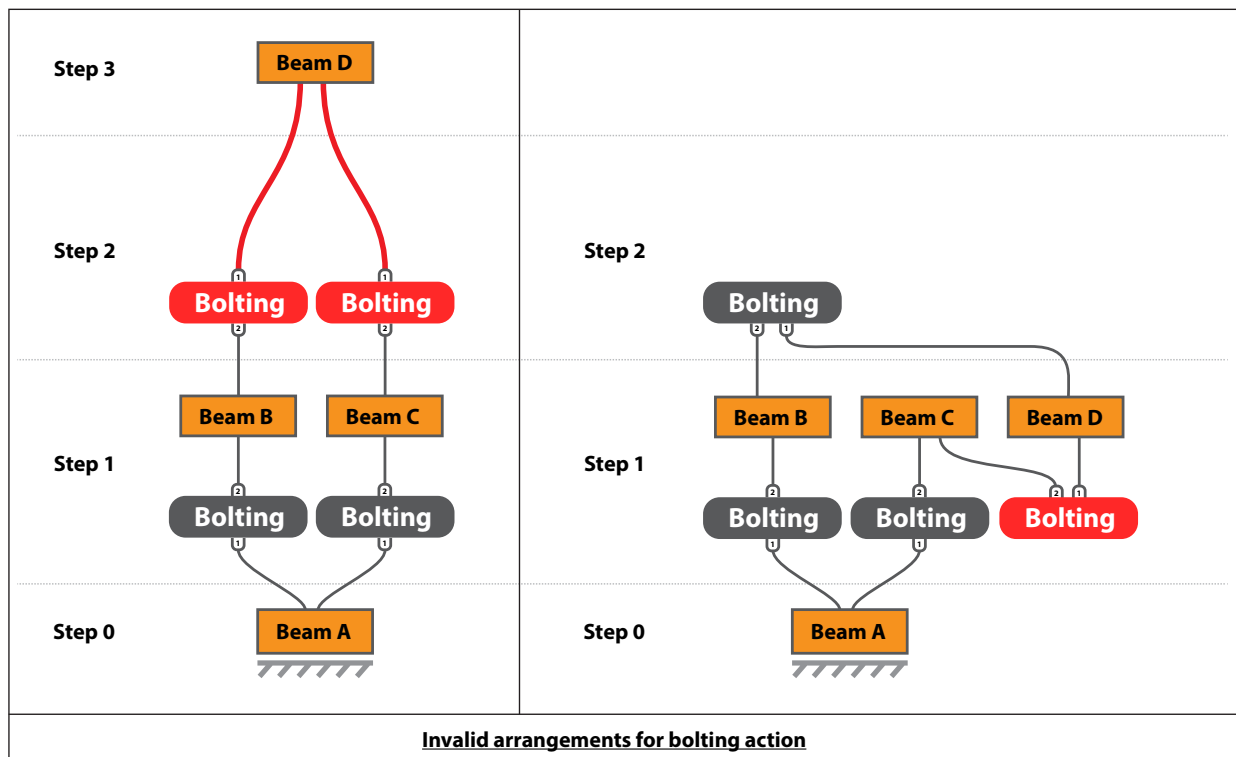
The designer can edit and rearrange the sequence of nodes in an assembly graph using the CAD plugin's graphical user interface. However, some of these arrangements may be invalid or problematic. It is possible to encode a "rule set" specific to each Action Node, such that the plugin can automatically detect and warn the designer about potential problems.

For example, a Bolting Action will have edges connecting to multiple Part Nodes. For the Bolting Action to be represented meaningfully, all of its connected Part Nodes should belong to the same step as the Bolting Action, or they should belong to an earlier step. In the example below (left), the two Bolting Actions (highlighted in red) in Step 2 are incorrectly connected to 'Beam D' in Step 3,

thus being invalid. The invalid connection is also highlighted in red.

For the animation algorithm to pick a non-moving part in the bolting movement, at least one Part Node must belong to an earlier step. In the example below (right), the Bolting Action (highlighted in red) in Step 1 is connected to 'Beam C' and 'Beam D' which are both in Step; thus violating this rule.

These conceptual relationship between Part Nodes and Action Nodes is generalizable to many types of assembly. When the need arises to develop a different type of Action Node, one should re-visit the set of properties that is specific to the action, the validation checks, the visual representation of the nodes in the graph, the animation logic and its global settings.



3D Model, part libraries and custom parts

Although this thesis does not deal with the creation of CAD models or CAD assemblies, it is important for these CAD models to be paired with

their properties for the CAD plugin to use them. This allows the plugin to extract not only the geometrical models but also associated properties such as name, colour and model number of the parts. The table below provides an overview of properties and how they are used.

Property	Relevance	How it is used
Displayed Name	All Parts	<ul style="list-style-type: none"> Web player provides a part list and automatically generated textural assembly instructions.
Category Name/ Subcategory Name	All Parts	<ul style="list-style-type: none"> Web player can provide a Bill of Material that is sorted by categories. Action nodes may use this information for validation.
Model Number	All Parts (only if displayed name is not specific enough)	<ul style="list-style-type: none"> The displayed name should be kept short or quick reading. Model number allow more specific name for Bill of Materials, sourcing, purchasing.
[list] URL of 3D model	All Parts	<ul style="list-style-type: none"> Web player retrieves the 3D model of each part from a given repository (list of URL for redundancy)
Assembly Tool (Pointer to a tool library)	Parts that require tools	<ul style="list-style-type: none"> Web player can provide a tool list and/or animate the tool.
Tool Position	Parts that require tools	<ul style="list-style-type: none"> Position information of where the tool should be located and animated relative to the part
URL of photo	All Parts	<ul style="list-style-type: none"> Web player can show a photo of a part to the user for identifying it
Information Message	Parts that require extra information	<ul style="list-style-type: none"> Web player can show information message about further assembly instructions
Warning Message	Parts that require warning	<ul style="list-style-type: none"> Web player can show a warning message about potential assembly mistakes

Some of these properties are specific only to a certain type of part. For example, “diameter” is relevant to a screw, but not applicable to a rectangular beam. The table below is an example of parameters relevant to a screw part.

Property	How it is used
Diameter	<ul style="list-style-type: none"> Web player can be developed to highlight all screws of the same diameter.
Length	<ul style="list-style-type: none"> When animating a bolting action, the bolt is inserted into the bolt stack. The length property determines the distance of this movement.

Many CAD platforms have already implemented some object associated properties such as part name, layer name or colour. Some of these

platforms or file formats even allow users to add custom property fields. However, there is currently no file format that supports custom properties

fields and is supported by major CAD platforms. In order to develop a plugin algorithm that can use the same model library between different CAD software, I suggest a pragmatic approach to link a separate data file to the part models.

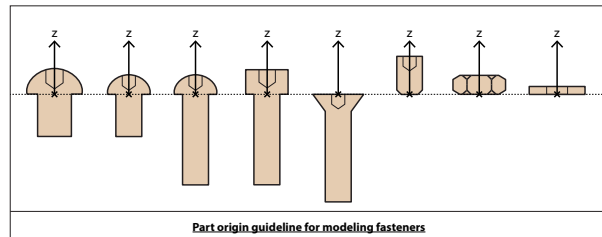
In the scenario where a part model is created by part manufacturers, this data file can be distributed together with the 3D model. When the machine designer imports the CAD model, the plugin can automatically attempt to read the associated data. These data are then copied and stored with the Part Node object. Designers can later choose to override the data for each individual instance of the part, without altering the data sheet.

Custom designed parts can be handled differently by the plugin and the exact method is specific to the CAD platform¹⁰. For example, if a designer designed a custom screw in a CAD platform using a specific “screw command”, it may be possible for the CAD Plugin to interrogate the properties of this screw using the CAD software’s API, thus obtaining the essential information. It is also possible to present users with a form to fill in these information.

Parts that have one or more rotational axes should be modelled such that their axes pass through the model frame’s origin point. This ensures naturally looking rotational movement during the interpolation. Further details will be explained in the later chapter “Web Player:

10 Because each CAD platform requires a separate implementation of the *Put it Together* CAD Plugin, it is possible to develop functionalities case by case.

Interpolating Movements”. The following diagram shows an example of how common fasteners should be modelled with respect to their origin point and their primary axis parallel to the Z axis¹¹.



3.2 CAD Plugin - for Rhino 5

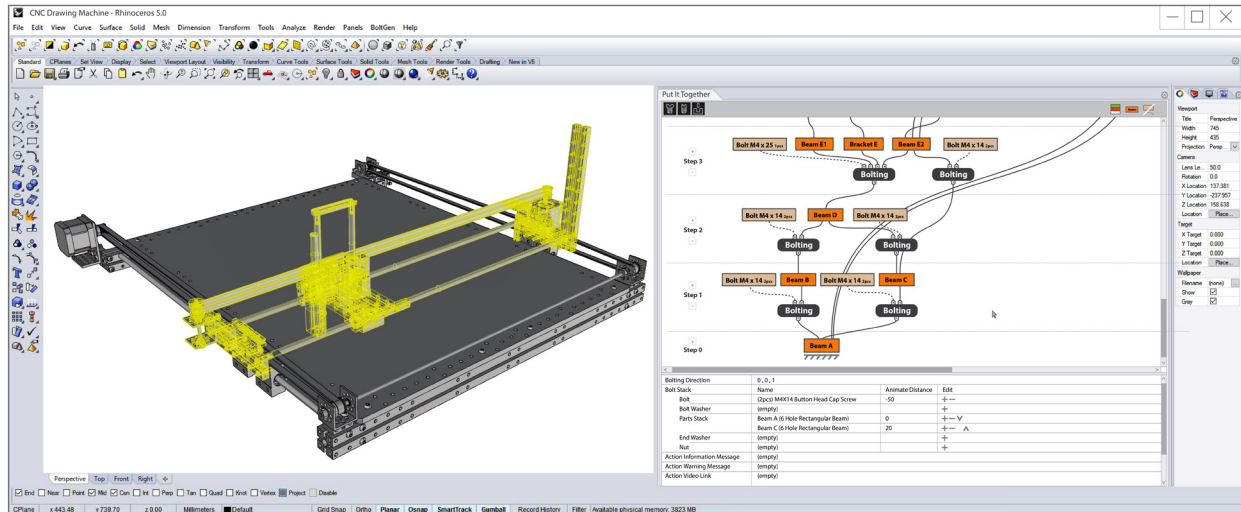
The CAD Plugin is a software that interfaces with the CAD package that designers use. It offers tools to create and edit the assembly graph and the properties of the nodes. This thesis encourages other developers to contribute in developing the CAD Plugin for different CAD platforms. This chapter describes the requirements and concerns for developing such plugins.

As part of the technical demonstration, I have developed a CAD Plugin for Rhinoceros 5.0 using VB.Net language. I implemented most of the assembly graph features but not the graphical user interface due to time constraints. This plugin allowed me to create test cases of assembling different machines. The test cases are used in later development of the Web Player which was shown to students in the educational workshops.

11 The reason to model the screws parallel to the Z axis is a convention such that the ‘Bolting Node’ can infer the bolting direction automatically. This convention is not a result of interpolation limitation.

Graph editing

The designer interacts with a Graphical User Interface of the CAD plugin to create and edit the graph. The image below is an example of the user interface, showing a canvas with a visual representation of the assembly graph and tools to edit the graph.

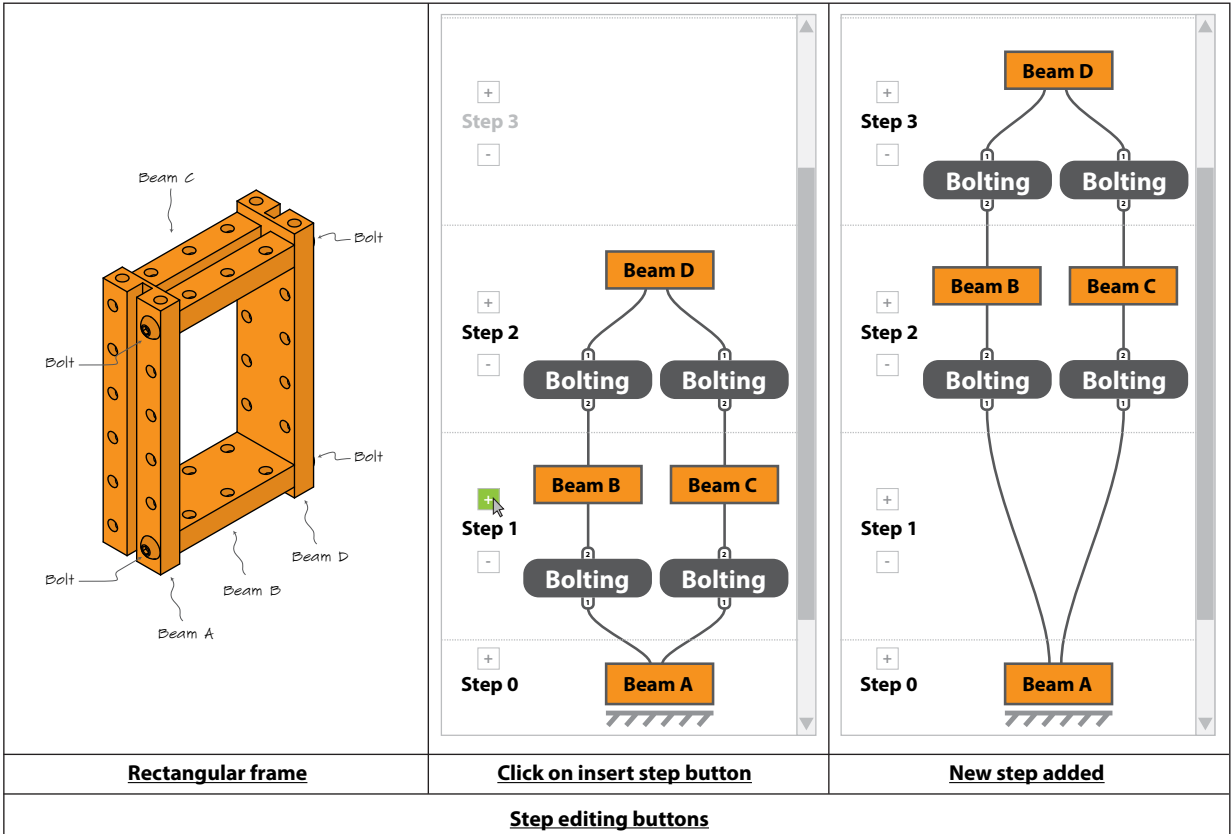
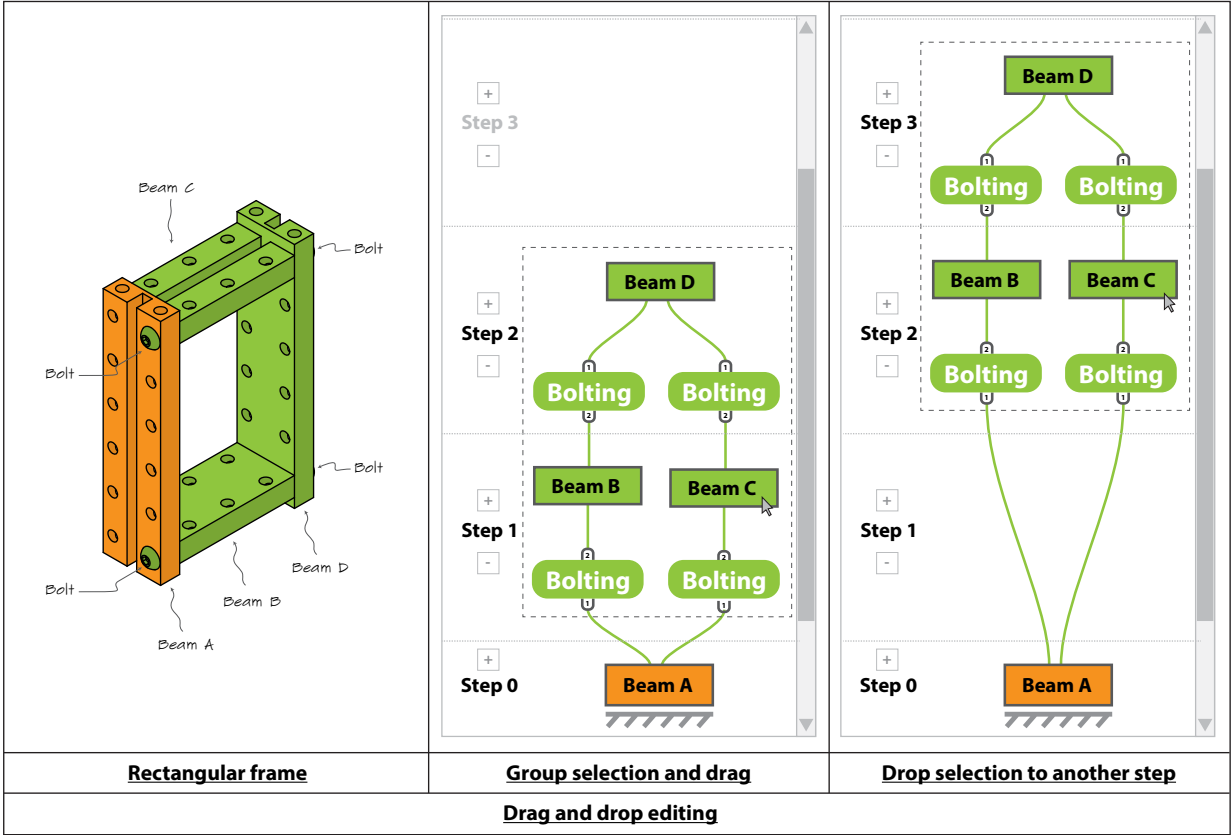


Example of the CAD Plugin

The Part Nodes are automatically created when the Action Nodes are linked to a part. The Action Nodes are created by clicking on one of the Action Node buttons and then selecting the related part models in the CAD environment.

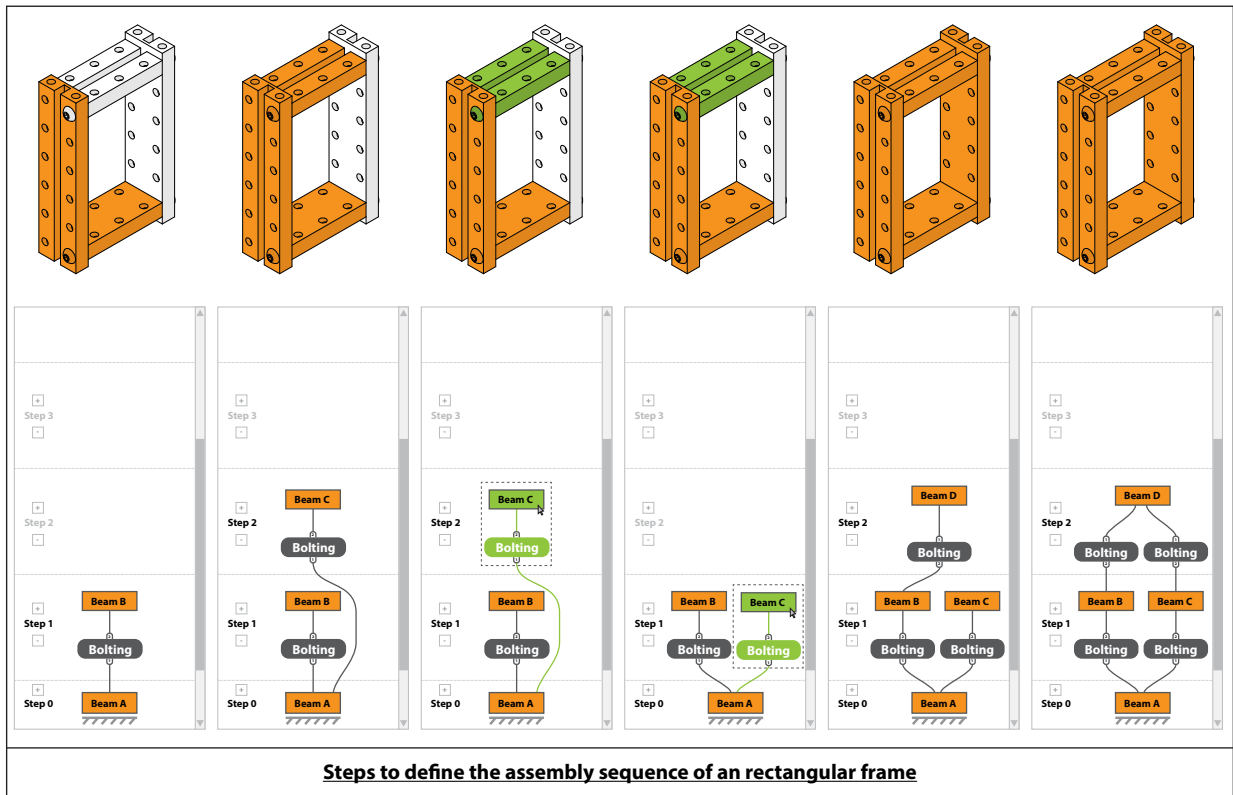
Designers can pan the canvas by holding the right mouse button while dragging; zoom in or out can be controlled by the mouse wheel. Node selection is possible on all nodes by the left mouse button click; designers can delete a node after selecting it. Multi-select is possible by Shift-Click or window lasso selection method. These mouse interactions are a generic guidelines and actual implementation should match the camera control system of the CAD platform.

The graph can be rearranged by mouse actions that select and drag the nodes into different vertical or horizontal positions. Vertical drag and drop should provide automatic snapping to assist a neat layout. The Insert Step Button and Remove Step button can be conveniently used to insert or remove a step in the middle of a long sequence. The example below shows how the Drag & Drop and Insert Step Button can be used to achieve the same result.



Various visualization modes can be implemented to assist the designer's workflow. (1) Hide / Show Dependent Parts, (2) Show Full Name / Common Name, (3) Tri Colour Visualization in CAD Environment. The tri-colour feature allows the parts in the model space to temporarily change their visualization colour, to show whether they have already been included in the graph (orange) or not included in the graph (grey) or included and is

currently selected (green). The following example shows the process of specifying the actions on a rectangular frame with 4 parts and 4 bolting actions. (In this example, (1) Dependent Parts are hidden, (2) Common Name is shown, and (3) Tri-colour visualization is turned on.) Notice the designer performed a drag and drop to rearrange the two Bolting Actions.



Persistent storage of the graph

The nodes and edges that make up the assembly graph is specific to a particular CAD assembly file. Depending on the extendibility of the CAD file format of a particular CAD platform, the CAD Plugin can either store the graph within the CAD file format or store it separately as a file alongside the CAD file. If a separate file is used, it is advised to keep the file name identical to the assembly

file. It is also ideal to trigger the saving action of the graph when the CAD document is saved.

In this thesis, an experimental CAD plugin is developed for Rhinoceros 5 in VB.net language. The native file format of Rhinoceros (.3dm) allows the plugin to store the graph internally as a custom "User Data" serialized into XML format.

3.3 From graph to animation

This thesis has limited the scope of animating machine parts to parts that are rigid bodies; thus reducing the complexity of animation to motion that are proper-rigid-transformations — a combination of only translation and rotation movement¹². This section discuss about algorithms that analyse the assembly graph and create keyframes for the animation.

Keyframes

Each Action Node in the assembly graph will translate into one short assembly animation. The Part Nodes connected to an Action Node are the parts involved in that animation. This animation is described with a group of keyframe containers that represent the motion of each involved part during the animation.

A separate keyframe timeline is created for each individual component and allows each to move independently. A timeline consists of at least two keyframes that describe a transformation from one position to another position, from a specific time to another specific time. Both **position and orientation** of a part (a transformation from the world frame to the model frame) are stored in the keyframe as a translation vector and a rotation quaternion. While translation vectors can easily be interpolated linearly, Gortler's book (Foundations

of 3D computer graphics 2012) provides the algorithms for interpolating quaternions smoothly.

While quaternions are ideal for producing naturally looking rotation movements, it is worth noting that that typical code implementation for quaternion interpolation will find the shortest distance between two orientations. Thus, it cannot create a rotation more than 180 degrees. For animations that require more than 180 degrees, the CAD Plugin should create intermediate keyframes to ensure a complete rotation.

Real time unit (seconds) is used to mark the position of a key frame. Different from many computer animation software which use frame numbers, real time rendered animations do not always have a constant frame rate. To ensure consistent motion in a variable frame rate, the exact position of the object in each frame is calculated based on the elapsed time when the frame is drawn.

Animating strategy

Each Action Node has its specific animating strategy. For example, inserting animation is a translation of a part, bolting animation requires translating and rotating the bolt at the same time and while simultaneously animating other parts in the bolt stack. Even the same Action Node can have multiple animating strategies. For example, a bolt can be animated simply by translating it into the nut; or it can have the more realistic helical movement. The exact animating strategy depends on the implementation of the CAD plugin and should reflect the need of the target audience. It is the CAD plugin's responsibility to compute these

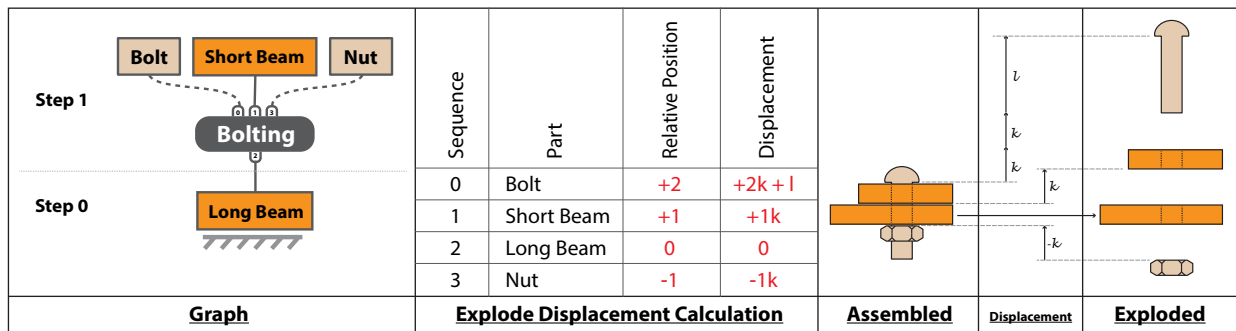
¹² Uniform and non-uniform scaling, skewing and mirroring are unlikely or impossible to happen to a rigid body during assembly. Think about the impossibility of shrinking a steel bolt.

keyframes from the properties in the Action Node and its connections to the Part Nodes.

The following diagram shows one of the possible animating strategies for a Bolting Action. The animation starts by showing an exploded bolt stack and then translating all components of the bolt stack towards the final assembled position. There are only two keyframes per object in this animation. The amount of translation in each part is calculated based on a global setting (k) and its position relative to a chosen static part. The **Rel-**

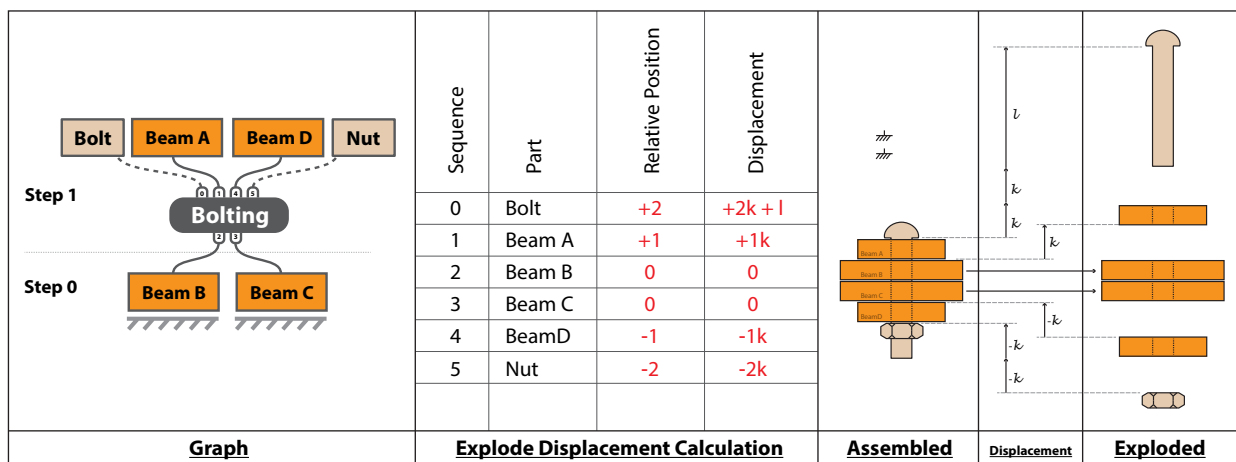
active Position is an integer (positive or negative) that starts from the Bolt, Bolt Washer, Parts in the Bolt Stack, End Washer, and Nut¹³. The parts that belong to the previous steps have a relative position of zero. The exception is the movement of the bolt, which also takes into account the bolt length (l).

¹³ Bolt Washer, End Washer and Nut are optional in defining the Bolting Action. Together with the Bolt, these are Dependent Parts of the Bolting Action. When multiple bolts are collapsed into one Bolting Action, all these dependent parts have to have the same number as the number of bolts.



The example below shows a more complex Bolting Action where two parts belong to a previous step. Both of their relative positions are

set to zero, which causes them not to move in the animation.



Note that there are many other ways to animate a bolt stack, section 5.10 describes how strategies can be designed to reflect human behaviour and provide a more illustrative approach in explaining the movements.

The function that computes the displacement value is specific to the Action Node and is developed by the developer of the CAD plugin. This function is invoked during the creation of the Action Node in the assembly graph. The computed displacement values are then stored in a table in the Action Node's property. They can be manually overridden by the designer if necessary.

Because it is possible to place multiple Action Nodes into one step and animate them together, and it is also possible for them to have different animation times (up to the Action Node's designer or user to override), the shortest animation will wait until the longest animation is completed before looping or proceeding to next step. Typically 3 seconds per translation is appropriate for the animation.

Encoding the animation

When the assembly graph is completed by the machine designer, an "Export" function can be triggered to compute all the motion keyframes and encode them in an animation description file. This file combines information from the part file's property and the movements of the animation in a readily useable format for the web player. This file is encoded in JavaScript Object Notation (JSON) and its schema is attached in the appendix. It contains metadata of the animation, descriptions of the parts used, descriptions of each

part instance in the finished assembly and a list of motion keyframes for each step. This file does not contain the 3D geometrical models of the parts, but contains a pointer to the repository of those files.

These decisions are made to optimize for web deployment where JSON can be readily used by the Web Player written in JavaScript. 3D models are loaded separately from an online repository to reduce redundant file exchange and enable offline caching of the models by the Web Player. In the context where hobbyist design, create and share different machine assemblies, it is very common for them to use similar parts but arranged in a different way; therefore, offline caching of part models can save significant bandwidth.

This thesis has envisioned the Web Player to be an open source software that can be developed by different developers and modified for different user interfaces or functionalities as appropriate for different applications. Thus, the animation description file is intentionally designed to be as extensible and flexible as possible to function across various permutations and implementations of the Web Player.

3.4 Web Player

The Web Player is the final step of the *Put It Together* tool chain. It is a 3D environment that can be accessed from a web browser, showing a real time rendered animation, and can be controlled by the viewer interactively. Its role is to automatically read the animation description

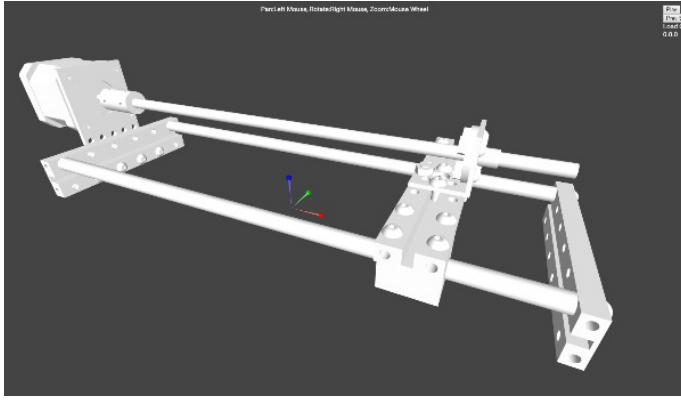
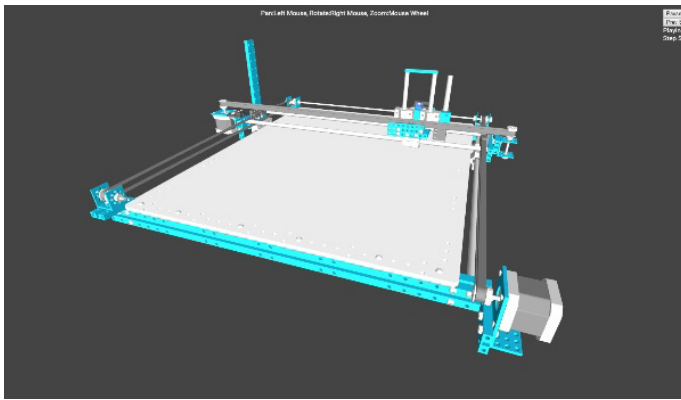
file, fetch the 3D models from repositories, and display the animation to the novice.

The goal of the Web Player is to present the assembly sequence in an interactive way most easily understood by the novice. Therefore, it features various user interface techniques that make it easy to learn and easy to use by the novice, thus reducing the overhead effort for learning how to use the interface itself.

This section provides an overview of the functions of the web player and technical development details. The next chapter, "Observation", discusses advanced features that are inspired from the

workshops that can be implemented in future works.

The Web Player implementation in this thesis underwent an iterative software development, through successive evaluations in the educational workshops that are described in the next chapter. While certain features described in this chapter constitute the basic implementation of the player, some features respond to the specific needs in the education setting of these workshops and may not be valid in all applications. In the development of this thesis, 5 iterations of the Web Player have been developed as incremental test to these features. The following table gives an overview of the key features tested.

	<p>Iteration 1:</p> <p>First test on Step by Step Animation controlled by two step-control buttons. World Coordinate Symbol is added to help orient the model.</p>
	<p>Iteration 2:</p> <p>Models are modelled with a texture that resembles the true colour of the components.</p>

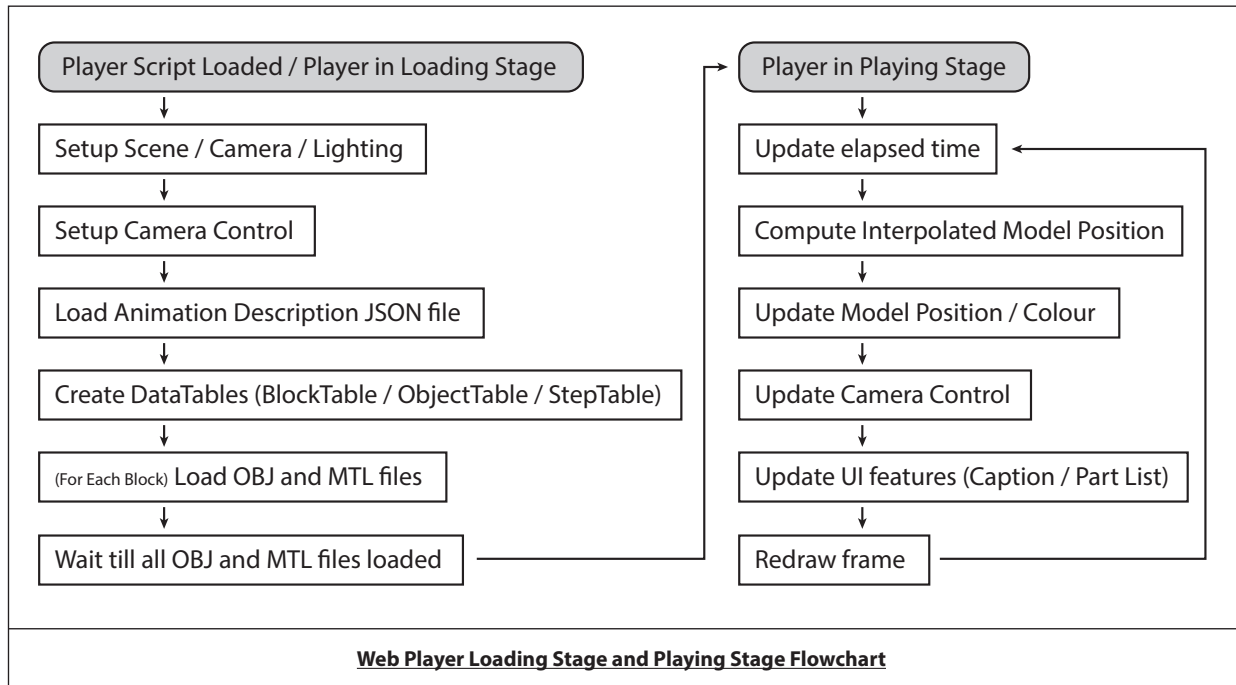
	<p>Iteration 3:</p> <p>Part list for each step are shown in a collapsible menu on top-left corner of the screen.</p>
	<p>Iteration 4:</p> <p>Automatically generated captions are added to the bottom of the screen to explain the action.</p>
	<p>Iteration 5:</p> <p>Camera automatically reposition and centers on the objects of interest at the beginning of each step.</p>

Virtual environment with Three.js

The core of the player is to provide a 3D environment where 3D models are placed and rendered for the user. This requires a framework to store the position and orientation of 3D models and functions to manipulate them. Three.js is a JavaScript library that provides most of these functionalities. It provides functions for importing and transforming 3D models in a virtual scene. It also wraps WebGL function calls that initiate the rendering process by the user's graphics hardware. It contains OrbitControl.js, which provides the mouse gesture functions for camera control.

The Web Player development in this thesis consists of an html page that hosts a series of JavaScript files which imports the Three.js libraries. CSS files are also included to control the visual appearance. During the “Initialization stage” of the player, the script sets up the 3D environment with

an empty scene, a virtual camera and various control functions. After all 3D models are loaded, the player enters and stays in a “Playing stage” where a number of functions update object positions and perform the redraw cycle. A brief workflow diagram is shown below.



Importing models

The part models described in the section 3.2 are exported in two file formats. OBJ for the 3D geometry (in mm units) and MTL for the material property. Both are plaintext-based open file format¹⁴ that are supported by many CAD platforms. They can be easily exported by those platforms

and can be easily imported by the OBJMTLloader.js in Three.js library.¹⁵

The current player implementation reads the Animation Description JSON file during the “Loading Stage” and creates a few data tables to assist the process of loading and placing the 3D models. A “Block Table” is created to represent all the unique

¹⁴ Open file format are formats that can be used free of charge.

¹⁵ One drawback is the large file size of OBJ and MTL format causing bottleneck in loading time. This should be addressed in future works on either compressed OBJ MTL or other file formats.

3D parts used¹⁶; each row in the table contains the link to the OBJ and MTL files. The software will create one asynchronous loading call per row to fetch the OBJ and MTL files. Once the files are fetched, they are stored as a Three.js Geometry representation.

Each part used in the animation is represented by a row in an “Object Table”. Even when some parts share the same 3D model (for example, many screws of the same geometry), they require their unique row in this Object Table to represent their unique locations and animation keyframes.

It should be noted that the complexity of these 3D models directly impairs the performance of the rendering process. The polygon count of the 3D mesh models should be reduced when designing for devices with slower graphic hardware¹⁷. In the pen plotter experiment (refer to the next chapter), the assembly have 224 parts with an average of 1.5k vertices and 1.7k polygons; the largest model has 13k vertices and 13k polygon. When all the parts are displayed (the last step is the most complex step), it was able to maintain 50 fps on a 2015 mid-range PC laptop¹⁸ and 20 fps on a 2015 mid-range Android phone¹⁹.

¹⁶ For example, many of the same type of screw (M4x16 Button Head Screw) is used in the assembly. Only one record of the screw will be found in the “Block Table”, but each records of its many instantiation will be recorded in the “Object Table”

¹⁷ Most complex 3D model we have tested contains 13k polygons.

¹⁸ CPU: Intel(R) Core(TM) i7-3630QM @ 2.40GHz, 2401 Mhz, GPU: GeForce GT 650M@745MHz, 2GB GDDR5

¹⁹ CPU: Quad-core 2.2 GHz Cortex-A17 & quad-core 1.7 GHz Cortex-A7, GPU PowerVR G6200

Camera, and camera control

A perspective projection provides a more natural appearance to novice users and conveys the depth of the scene better than parallel projection; therefore, a 60 degree field of view²⁰ perspective camera is used in this thesis. Near plane is set at 1 and far plane is set at 10000. Units are in millimetres.

Three.js camera uses a right hand coordinate system, which is similar to most CAD systems. By the conventions of computer graphics, objects on the XY plane are drawn on the screen and the Z axis conceptually points out of the screen. However, with the convention of machine design, it is common for objects to be modelled where the Z axis points to the top of the machine or towards the sky. And it is more natural to display these 3D models where its “Up” direction (Z axis) corresponds to the top of a computer screen. Therefore, changing the “Up” setting of the perspective camera to “Z axis”, can result in a default camera that shows a more natural side view of the machine, instead of the top view.

Users can orbit the camera by first holding down the right mouse button, and then dragging the mouse. This dragging motion translates to the camera orbiting around an imaginary center point while the Z axis of the model space will always point towards the top of the screen. This Z axis lock is intentionally programmed to prevent the

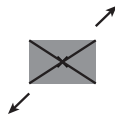
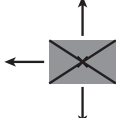
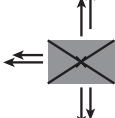


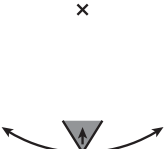


²⁰ Three.js perspective camera accounts for the field of view based on the width of the screen only. Special care is needed when developing for vertical screen devices.

user from disorientation, such as when using a trackball control.

it is very close to the imaginary point. A potential workaround is described in next chapter.

The mouse wheel controls the dolly of the camera, so that users can magnify the object of interest. Scrolling the mouse wheel moves the camera towards or away from the imaginary point while the field of view (FOV) is constant. Even though this is commonly referred to as “zoom”, dollying is different from the concept of zooming in photography where FOV changes. One down side of the dolly control is the saturation of the dolly when

Users can also control the panning of the camera by first holding down the left mouse button and then dragging the mouse. This translates both the imaginary orbiting point, camera position and camera target position. It is worth noting that novice users generally have difficulty grasping and differentiating the concept of orbiting and panning the camera. Two suggestions are made in the next chapter to reduce the need for panning.

Camera Control	Dolly	Orbit	Pan	Walk
Front View				
Top View				
Center of Orbit Movement	Stationary	Stationary	Parallel to Screen	Normal to Screen
Different types of Camera Control				

The above diagram shows the difference between the three camera controls. The black cross is the imaginary orbit center.

Alternatively, users can also use their trackpad or touch screen to control the viewing angle. In the case of touch screen control, single-finger-drag-gesture orbits the camera, two-finger-pinch-gesture dollies the camera and three-finger-drag-gesture pans the camera.

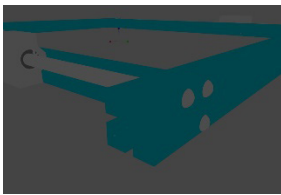
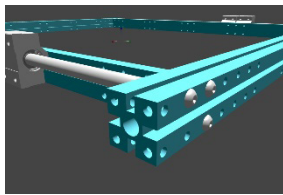
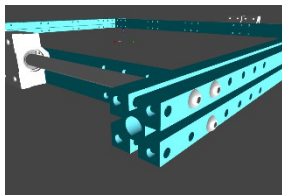
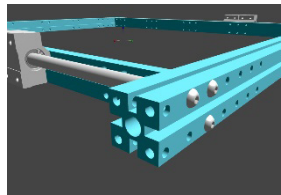
Light

One ambient light is added to the scene and one directional light is added to cast shadows on the 3D model. Both lights are light grey to avoid over exposing the colour channel during the rendering pass.

The directional light is set at an angle relative to the camera, to create different shades on different face normals. Without this directional light, all faces of an object will have the same shading intensity and it will be impossible to distinguish the 3D shapes. A comparison chart is included below to show how the directional light is angled relative to the camera to create a more identifiable shading.

This directional light is set to follow the camera position and orientation, such that the differential shading effect is maintained at all viewing angles. When the camera angle is changed by the user's gesture, the shade of the objects changes at the same time. This dynamic shadow help users understand the 3D shape better when they rotate the camera around the object. This is similar to the effect of rotating an object in the real world while the lighting condition and eye position are fixed.

Other computer graphic researchers have dedicated efforts into Non-Photographic Rendering (NPR) techniques that are more suitable for technical illustrations than photorealistic renderings. The implementation of NPR techniques are outside the scope of this thesis, but a summary of illustration techniques can be found in the paper by Gooch et al. (1998).

Ambient Light	Ambient Light + Directional Light (top)	Ambient Light + Directional Light (right)	Ambient Light + Directional Light (top right)
			
Flat shade	Top face is brighter but left and right faces have similar shade.	Right face is brighter but top and left face have similar shade	All faces have different shades. Top face has the brightest shade.

Material and shading

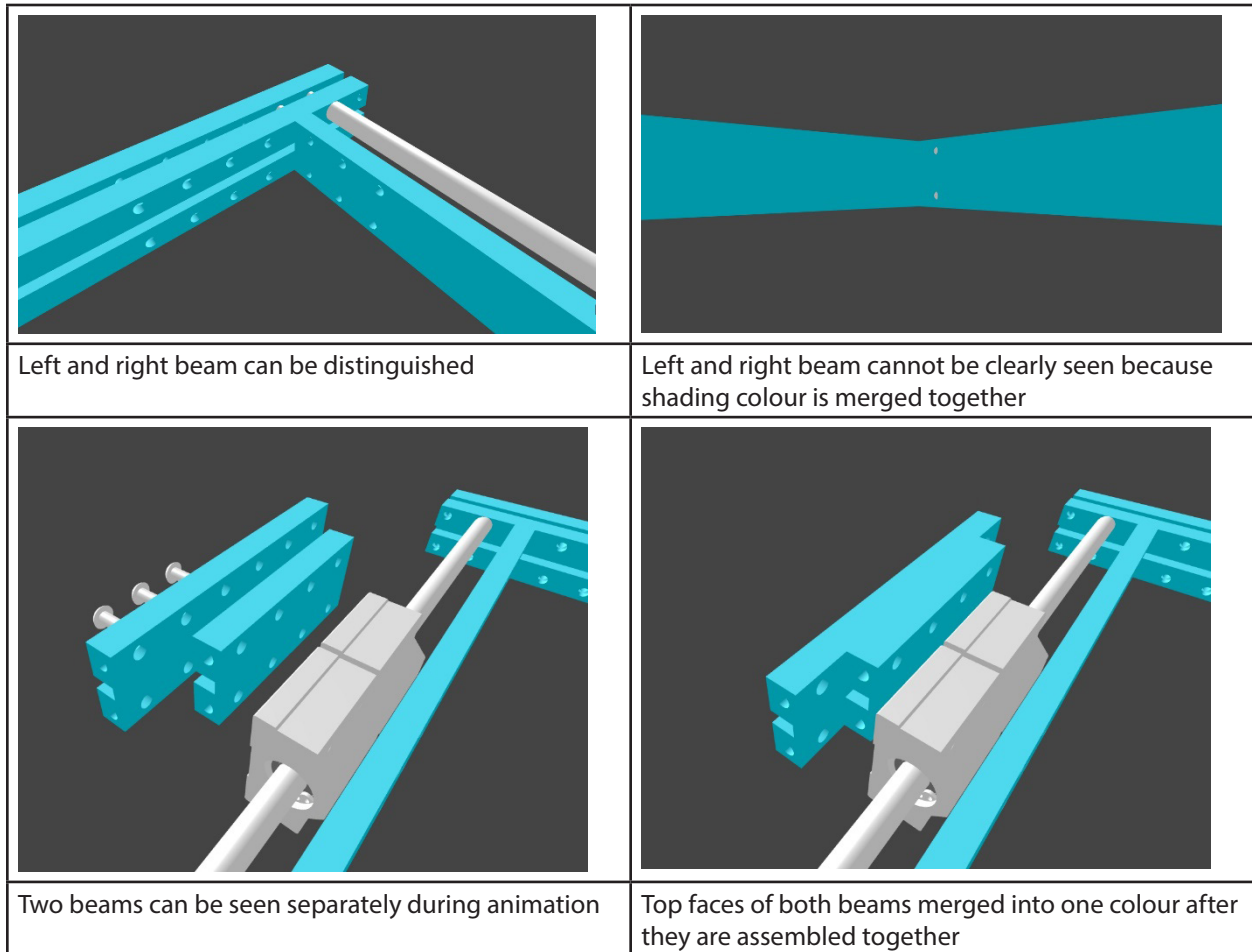
The 3D models rendered in this thesis are given a Lambertian material that simulates matte surfaces. This avoids the distracting artefacts caused by specular highlights. The diffuse colour is stored in

the MTL file which can represent different colours for each individual mesh face.

In some special cases, when two objects are touching each other, tangential surfaces may be rendered with the same shading. Or in another

case, where the orientation of different face normals have the same angle to the light source, the shading will also be the same. Both of these will cause confusion as the faces appear merged into one single face. It is therefore desirable to include

a silhouette outline to the rendered objects in the scene to clarify the shape. A shader based implementation of silhouette can be found in Card and Mitchell's paper. (2002)



Silhouette provides definition and separation between different objects and crease lines add definition to augment the shading. It is generally good to include one or both methods in the animation if hardware performance permits. There are different rendering techniques that can produce aesthetically pleasing silhouette or crease lines. Most of them use a custom GPU shader or multi pass rendering. Refer to the book by Gooch

and Gooch (2001) for various non-photorealistic rendering techniques and their implementation.

Real time rendering

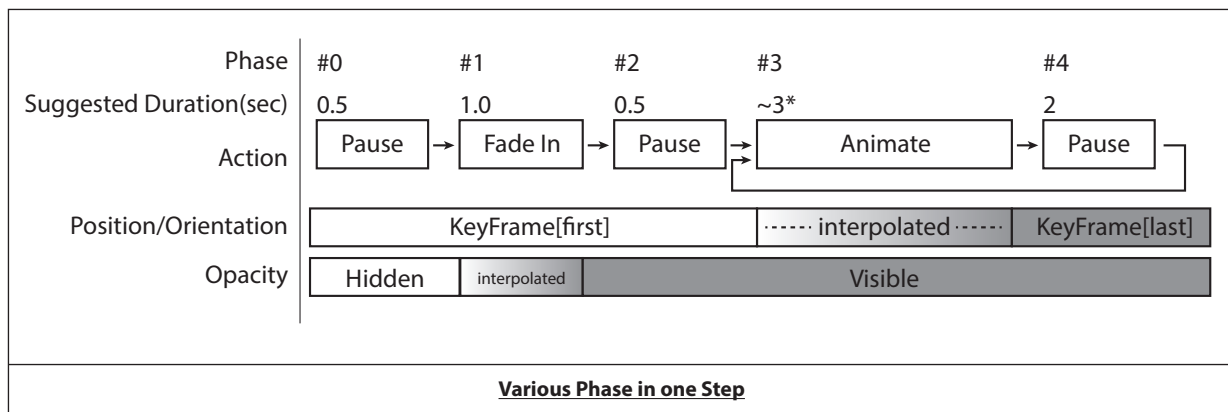
It is worth noting that both animated motion and interactivity are illusions created by the ability to redraw the scene with a high framerate — such that objects and camera appear to be moving in a continuous motion to the user. In order to animate a motion, the framework updates the position of the 3D models at every frame redraw. For the interactive camera control, user inputs are processed by OrbitControl.js at every frame and the camera positions are updated, such that the viewing position appears to be an interactive view, controlled by the user.

It is ideal to redraw the image at 60 frames per second (fps). However, due to hardware limitation and load sharing with other applications, it is not

always possible to achieve this frame rate. Therefore, it is best not to hardcode this frame rate and utilize the `requestAnimationFrame()` function which is supported by most browsers through JavaScript. This allows the browser to govern the redrawing speed and optimize for performance.

Interpolating movements

The Web Player designed for this thesis implemented a rather sophisticated animation sequence. Even though there could be as few as two keyframes describing one assembly step, the animation consists of 5 phases to improve understandability for the novice. The following graph shows the action and suggested duration of each phase. Notice the animation will loop phase #3 and #4 indefinitely until user moves on to another step.



* Actual animating time for Phase #3 is determined by the keyframes and may not always be 3 seconds.

The player keeps track of three variables related to this playback: Step, Phase and Time. Step, refers to the step of the assembly. This comes from the vertical levels in the assembly graph. The player by default starts at Step 1 and is then controlled

by the user using two buttons on the interface to move forward or backward. Phase²¹, describes

²¹ Animation Phase is an optional implementation for the player. It may not be relevant in all applications.

different phases when the animation is played (an optional feature in this specific implementation).

Time, is the current time of the animation. When the browser requests a redraw (typically at 60 frames per second), the value of time is updated automatically to integrate the elapsed time (Δt) since the last redraw. The player recalculates the current position, orientation and visibility of each object based on values of Step, Phase and Time. The value of time will be reset to zero at the end of each phase or when user advances to the next step.

Phase #3 is the key step for interpolating the position and orientation of the 3D model to create the animation. Each keyframe stores the position as a vector and orientation as a quaternion of the model frame relative to the world frame. Quaternion is chosen to be an unambiguous description of the orientation for interpolation. The position of the object at any instant between two keyframes is linearly interpolated, while the quaternions are spherically interpolated. Implementation and theoretical background of interpolating quaternions can be found in Gortler's book (2012).

It should also be noted that such interpolation results in a natural looking rotation movement only if the rotation axis goes through the origin of the model frame. For rotationally symmetric components such as shafts and screws, it is best to model their meaningful center point at the model frame's origin. For parts that cannot avoid rotation around an arbitrary center point, care should be taken to generate enough key-frames to maintain a visually meaningful trajectory.

4.1 Assembly task

4.2 Instructions (Variables)

4.3 Observables

4

CNC Workshops for Novices - Learning Experiment

In order to reveal issues and implications from people using the Web Player, I designed a series of educational workshops as usability tests. These workshops served as a demonstration and validation for *Put It Together* to function in an educational environment.

Eight workshops were conducted with university level students. The first three were pilot workshops taught using traditional methods such as paper instructions and static 3D models. They revealed problems in those methods and outlined the requirements for the Web Player.

Five more workshops were conducted using the Web Player described in Section 3.4 to provide feedback along with development of the player. These workshops were designed to be educational and attractive for university level students with no prior background in mechanical engineering. They contained a lecture about the fundamentals of CNC controlled machines and a hands-on assembly task to assemble a 2-axis pen plotter. Students were then given time to play with the assembled pen plotter to make drawings of their

own design²². Each workshop was 8 hours long, spanned between one or two days. The majority of the students came from MIT's undergraduate and graduate population. All students volunteered to provide feedback about the software.

Makeblock²³ sponsored 6 sets of the mechanical parts used in these workshops and these were re-used for each workshop. I sponsored the consumable materials such as drawing paper and drawing pens. Workshops were free for the students to attend.

²² The drawings that students made are designed by the students using tools I provided. Teaching students how to use these tools and operate the machine is outside the scope of this thesis.

²³ Makeblock is a Chinese company that makes Robotic Parts for DIY robot building.

#	Dates	Location	Series	Students
1	2015/01/12-16	The University of Hong Kong	HKU Winter Workshop	12
2	2015/07/20-30	City University of Hong Kong	AA Visiting School	10
3	2016/01/11-15	Massachusetts Institute of Technology	MIT IAP Class	12
4	2016/02/06	Massachusetts Institute of Technology	Weekend Workshop	5
5	2016/02/13-14	Massachusetts Institute of Technology	Weekend Workshop	10
6	2016/02/20	Massachusetts Institute of Technology	Weekend Workshop	5
7	2016/02/27-28	Massachusetts Institute of Technology	Weekend Workshop	4
8	2016/03/06	Massachusetts Institute of Technology	Weekend Workshop (DCG) ¹	4

* Workshop #1 was co-taught with Diego Pinochet

* Workshop # 3 was assisted by Julia Litman-Cleper and Mitchell Gu

香港大學建築學系
 DEPARTMENT OF ARCHITECTURE
WINTER 2014-15 WORKSHOP SERIES
 THE UNIVERSITY OF HONG KONG

TIME:
 2015.01.12 - 16
 10:30 - 17:30
VENUE:
 TBC, Knowles Building

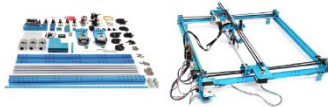
ELIGIBILITY:
 BA(AS) & MArch Students
REGISTRATION:
 2014.12.12 - 26

CONSTRUCTING NUMERICAL CONTROL

Computer Numerical Controlled (CNC) machines are everywhere. Since the invention of modern CNC mills at MIT in 1952, engineers motorized different "End Effectors" and invented CNC laser cutting, welding, plasma cutting, bending, spinning, punching, gluing, sewing, tape and fiber placement, routing, picking and placing.

Students operate laser cutters with ease every day, yet very few know how CNC technology works. This workshop will present the mechanical and electrical libraries about every component in a typical 2 axis CNC machine (i.e. a laser cutter). Participants (either alone or in groups of two) will then build their own XY axis CNC pen plotter in a hands-on session led by tutors from HKU and MIT. Students will also program their machines to make drawings that will be exhibited. Students are encouraged to come up with ideas to modify the machine for other novel applications, such as CNC milling, rotary axis, paint brush etc.

Sponsored by **Makeblock**
 Make Your Ideas Real!



Contact: Victor Leung vleung@hku.hk for course content enquiry.
 Registration opened to all BA(AS) and MArch students please email Agnes Cho choag@hku.hk with name and year of study. If you are interested in more than one workshop, please indicate your preference.

THE UNIVERSITY OF HONG KONG
 Faculty of Architecture 建築學系
 Department of Architecture 建築學系



IAP 11-15 January Build a CNC Drawing Machine

Computer Numerical Controlled (CNC) machines are everywhere. Since the invention of the first CNC milling machine at MIT in 1952, engineers are able to automate different "tools" with speed and precision, i.e. CNC laser and plasma cutting, milling and turning, welding by robotic arms, sheet metal bending, wire/cable punching, fiber placement, assembly line picking and placing.

This workshop is for design students with no prior knowledge of robotics. It combines theory lectures with lab sessions to introduce the mechanical and electrical systems in a typical CNC machine, i.e. a laser cutter. Students will build a 2 axis CNC pen plotter and program the machine to make drawings that will be exhibited. Students spend the second half of the course perfecting a drawing or modifying the machine for other applications, such as drawing an environmental action, using non-conventional features in cutting tools.

11-15 January, 2016 | 10a - 5p
All MIT students
 Engineering knowledge not necessary
 Quota: 12
Registration: <https://goo.gl/3NAmkl>
 Register by 25 December 2015
Enquiry: Victor Leung



Instructor **Victor Leung**
yleung0152@mit.edu
 SWArch's student in MIT Design Computation Group. Experienced in parametric associative CAD/CAM technology and wood fabrication. Interested in demystifying electronics and robotics for designers. Previously taught workshops in HKU and AA Vinting School.

Sponsored by **Makeblock** **MIT-SUTD Collaboration** **MIT Massachusetts Institute of Technology**

Poster of workshop #1

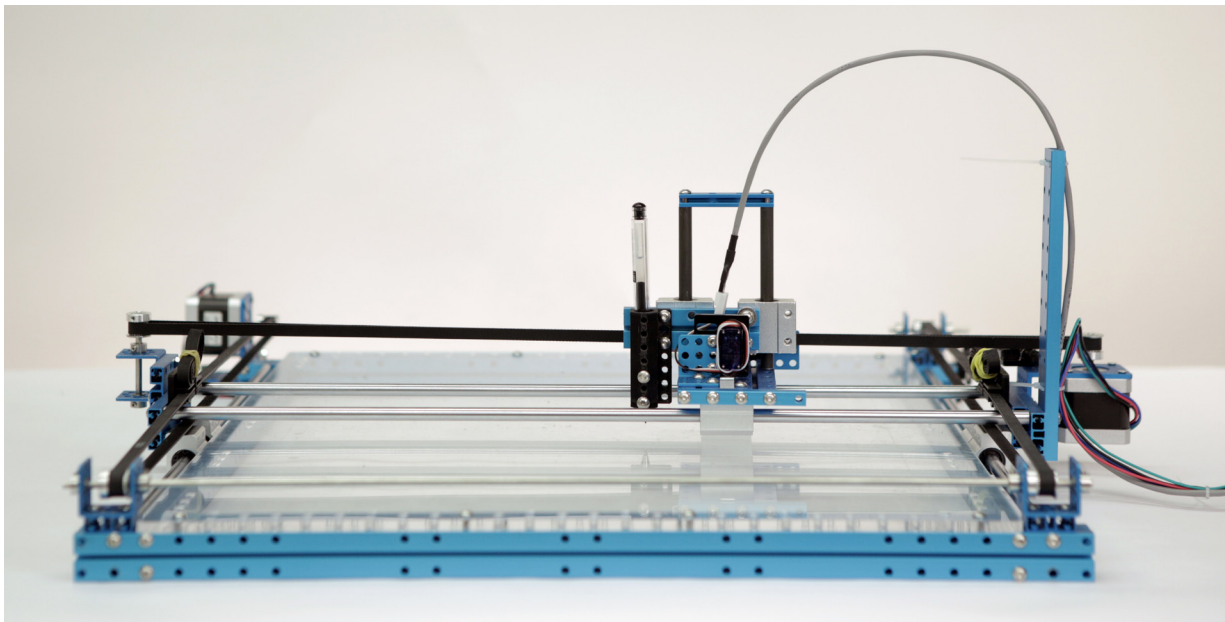
Poster of workshop #3

4.1 Assembly task

The students were asked to assemble a pen plotter as part of learning how to design and make CNC machines. A pen plotter was chosen because of its moderate complexity, covering three linear axis and different framing and motion transmission components. It contains many bolting and inserting actions that are relevant to general mechanical assembly. It required three different types of screw drivers and a wrench to

assemble. The assembly task took an average novice 3.5 hours to complete (or 3 hours in teams of 2 students). The method of instruction is described in the next section.

The photo below shows the pen plotter I designed using mostly Makeblock components and a few custom made components. The assembly task did not require students to design anything.



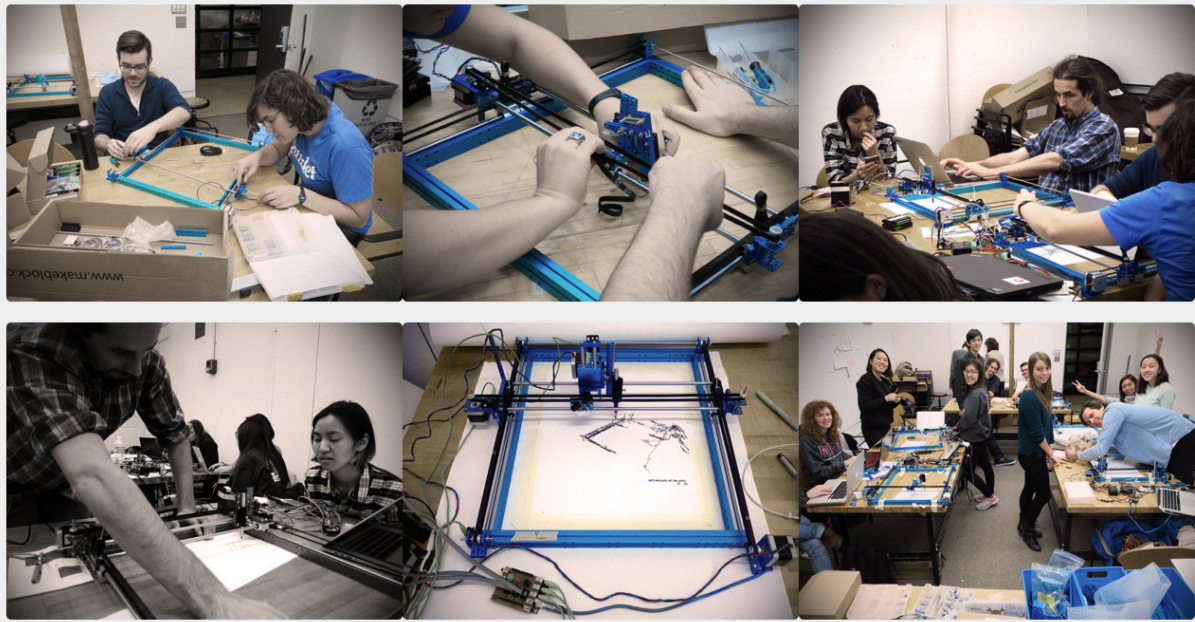
Λ Pen plotter assembled by students

4.2 Instructions (Variables)

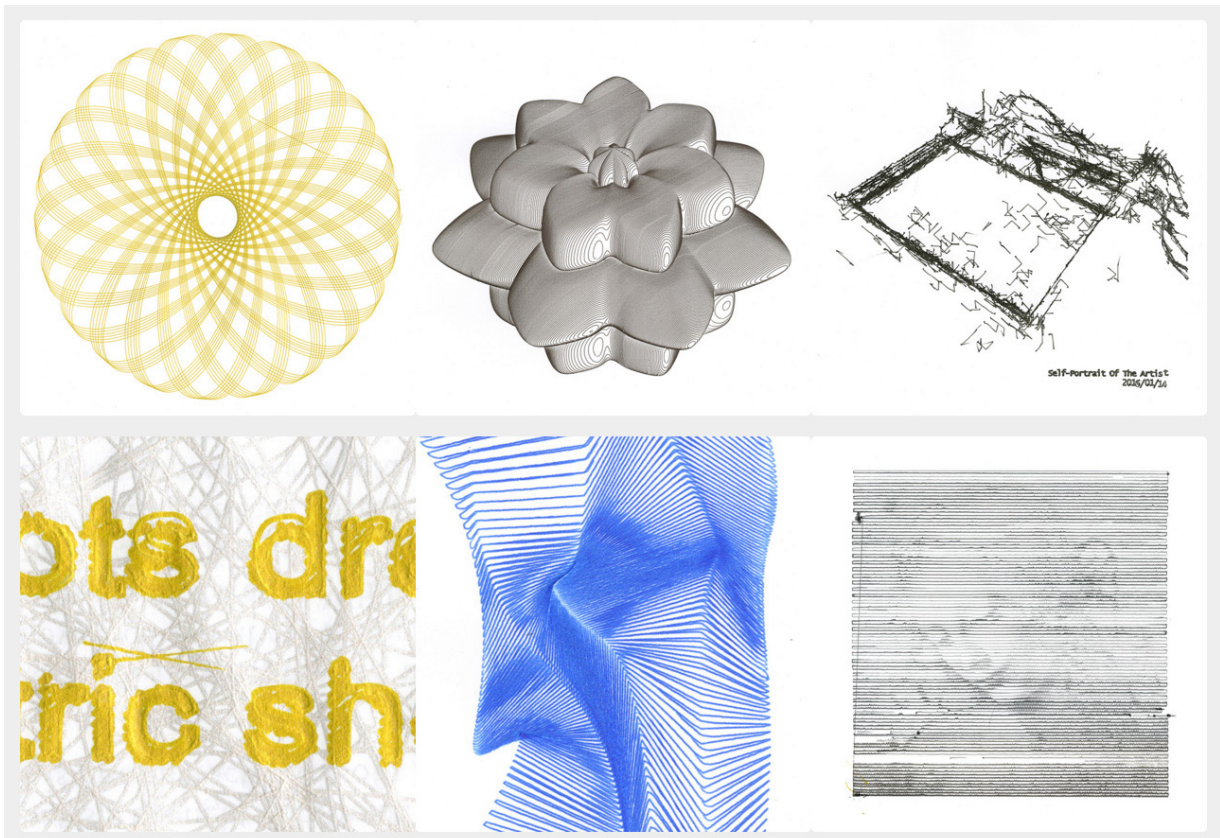
In each workshop (#3 to #8)²⁴, students were given the same task to assemble, but the instruction method was different in each workshop. The highlight of each workshops' goal is given in the following table:

Workshop #	Instruction	Intent
Workshop 1	Paper instructions prepared by Makeblock. Makeblock-designed plotter.	Reveal problems of paper instructions.
Workshop 2	3D model (in Rhino 5) of a large machine. Parts do not have names or realistic colour.	Observe student's behaviour when only a static model is given.
Workshop 3	3D model (in Rhino 5). Parts in the model contain their part names. Parts do not have realistic colour. Makeblock-designed plotter.	Reveal problems from the most common type of 3D model sharing method available online.
Workshop 4	Method same as above but with my Pen Plotter design. The following workshops will be in the same plotter design.	Intention same as above but act as control experiment for the following workshops.
Workshop 5	First use of <i>Put It Together</i> Web Player	Observe problems associated with the player.
Workshop 6	Added itemized part list for each step and realistic part color	Observe if part list help identify screw lengths. Observe if realistic color is confusing.
Workshop 7	Added captions for each step	Observe if captions helped assembly.
Workshop 8	Added automatic view zooming for each step.	Observe if automatic zooming help reduce time in finding parts.

²⁴ Workshop #1 and #2 is taught using a different version of the pen plotter. The later plotter design, is modified from the earlier one for better drawing quality, so that students can make better drawings.



^ Snapshot during the workshop



^ Drawings made by students with the XY Plotter.

4.3 Observables

The success of the Web Player depends on the criteria we use to evaluate it. It is similar to asking what constitutes a good teacher — it depends. In some applications, speed of understanding the step is important. In some, unambiguity and clarity is important. In some, mistakes should be avoided. And in some, it needs to be fun and educational.

This thesis is interested in observing the mistakes that students make. For example, assembling the wrong component, in the wrong direction, using the wrong tool, in the wrong sequence, in the wrong place, or completely forgetting to assemble a part. By observing these mistakes, we can draw conclusion on how the instruction method or Web Player can be improved towards different goals, without limiting ourselves to a certain type of application. It is also useful to analyse **the questions that students ask** the instructor, because these questions reveal the information that is missing or communicated ineffectively.

A questionnaire was also given to students to gather their opinions on the medium. A sample questionnaire is attached in the Appendix.

5.1 Basic knowledge

5.2 Identifying parts

5.3 Play / Pause mode

5.4 Occlusion

5.5 User interface navigation

5.6 No mouse

5.7 Showing alignment

5.8 Hard-to-animate steps

5.9 Calibration and other non-assembling step

5.10 How do people actually assemble a bolt stack

5.11 Sub-assembly

5

Observations and Interpretations

5.1 Basic knowledge

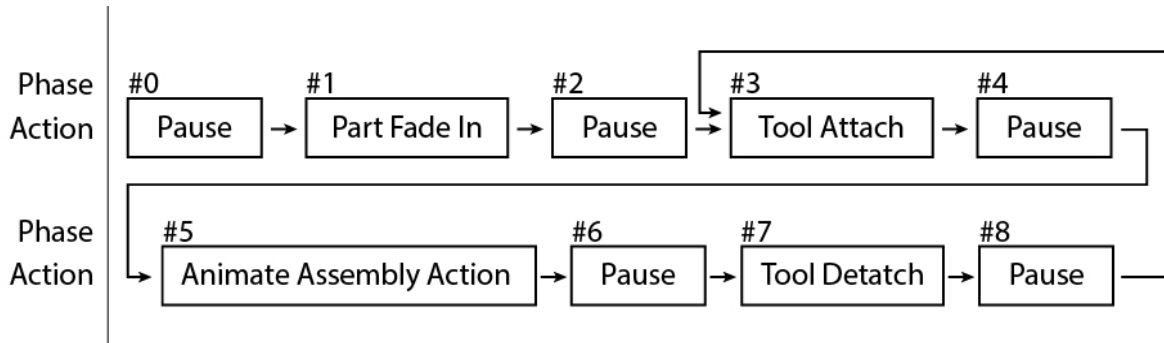
The design of the Web Player in this thesis has a hypothetical audience in mind. It is imagined to be undergraduate level students without engineering backgrounds who are interested to learn how to assemble a CNC machine. I have made these assumptions to estimate the audience's knowledge and capability. For example given two screwdrivers of different hex head, the person should be able to pick the right one by looking at the screw and the screwdriver. Or for example, if you tighten a bolt stack with a nut at the end, you need to use a wrench together with the screwdriver. However, observation from the workshops proves that even with careful consideration, there

can be a lot of variation between students. The mistakes that they made are often interesting and point to usability problems that are often overlooked.

One typical problem is not knowing how to use a tool. There are numerous instances where students confuse the tightening and loosening direction of their screws. I also found that it is not obvious to some people, that both the long and short arm of an L-shaped Hex key can be used for different clearances. The current Web Player implementation does not animate the location of the tool during the assembly (mainly due to the messiness of many tools obscuring the scene when there are multiple screws in a step). How-

ever, it is possible to **animate the tool** together with the part. This will require redesigning the animation phases of the web player, preferably with an extra step to show the attachment of the

tool, and another step to show the release of the tool. Below is an example of the animation flow in 8 phases.



Animation phases showing tools

It is worth noting the tedious process involved for the designer to select the right tool. For example a hex-head cap screw can be tightened by a hex screwdriver, by the long arm of the Hex key, or by the short arm of the Hex key. It can also be finger-tightened. However, the reward of this process allows the designer to visualize potential difficulty in the assembly process, such as a tight ratcheting window and unreachable location by hand.

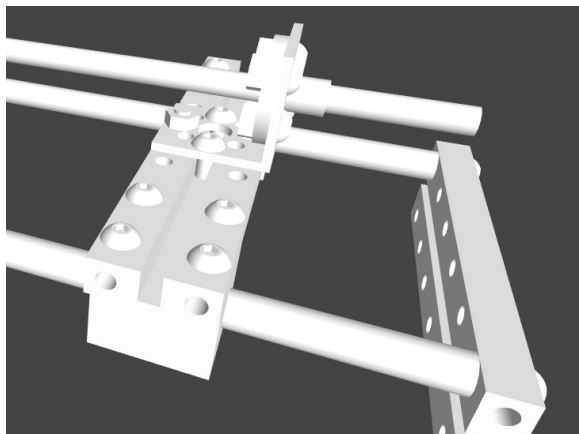
For people who have never assembled things with screws, it is common for them to over- or under-tighten a screw. Too loose and the machine falls apart; too tight and the threads or the parts are broken. Among other fascinating issues, I found that it is extremely difficult to communicate **“how tight is tight enough”**. Even a live demonstration is not possible to show the amount of torque that is felt by the hand. Without going into professional tools such as torque wrench, I have experimented with two ways to overcome this challenge. First is an undo approach: Let the

student tighten a few joints, and the teacher tries to tighten or loosen it and feel if the tightness is correct. The second method only requires a demonstration, I give a verbal instruction like this: “Tighten it hard enough so that none of the elements in your bolt stack can move independently from each other, test them by pushing them apart as hard as you can. (I show them how I use my fingers to push and where to push) For Tee joints, bend them slowly and observe if the joint opens a gap (I show them what a gap looks like).” Both methods are effective, but the second one does not require the presence of the instructor, the showing part can be captured in a video and distributed.

These basic knowledge assumptions are not trivial in the design process of the Web Player for a specific application. For example the audience watching a furniture assembly animation made by a furniture company is very different from a car mechanic watching a repair instruction made by a car company.

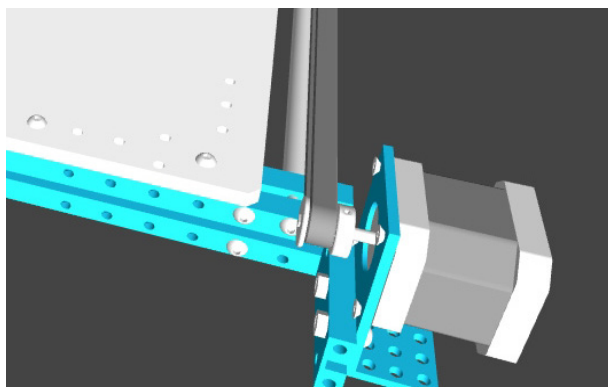
5.2 Identifying parts

One of the key mental processes for user when watching the instructions is to identify the part being animated on the screen. I tested a number of ways to do this and their effects are recorded below.



Λ 3D models with only grey colour

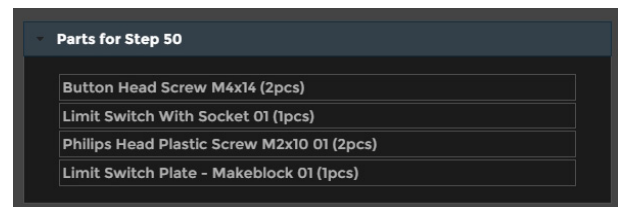
The first attempt was to use parts that had a generic grey colour, experimenting if the students can recognize the parts purely by the geometry. However, there was considerable feedback from students asking for a model that is coloured with the part's realistic colour, to help with identification.



Λ 3D models with real colour

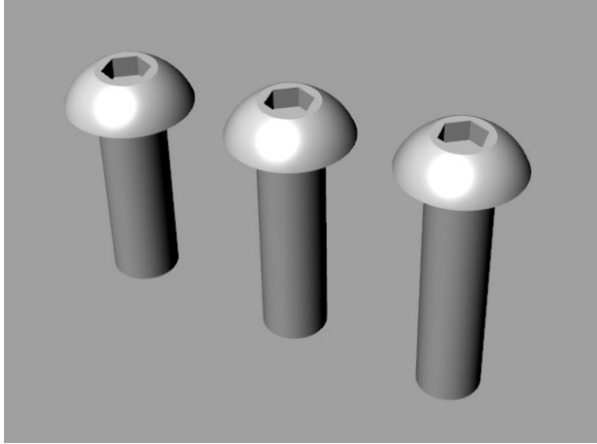
The second revision of the web player incorporated realistic colour. The MTL file allows each individual mesh face to have a unique colour, reflecting the realistic colour of the part. This worked well in the workshops and substantially reduced students asking “Is this the right part?”

However, for products that have different colour variation but not functional difference, the colour on the screen might be confusing to the user who has a different colour at hand. One possible mitigation is to produce a different animation for each product colour. Another possibility is to ask users to pick their product's colour before the start of the animation (applicable to furniture). Another possibility is to provide more information in a separate panel or as a warning dialog for the user, showing the alternative colour (applicable in DIY machines where parts may come from different manufacturer).



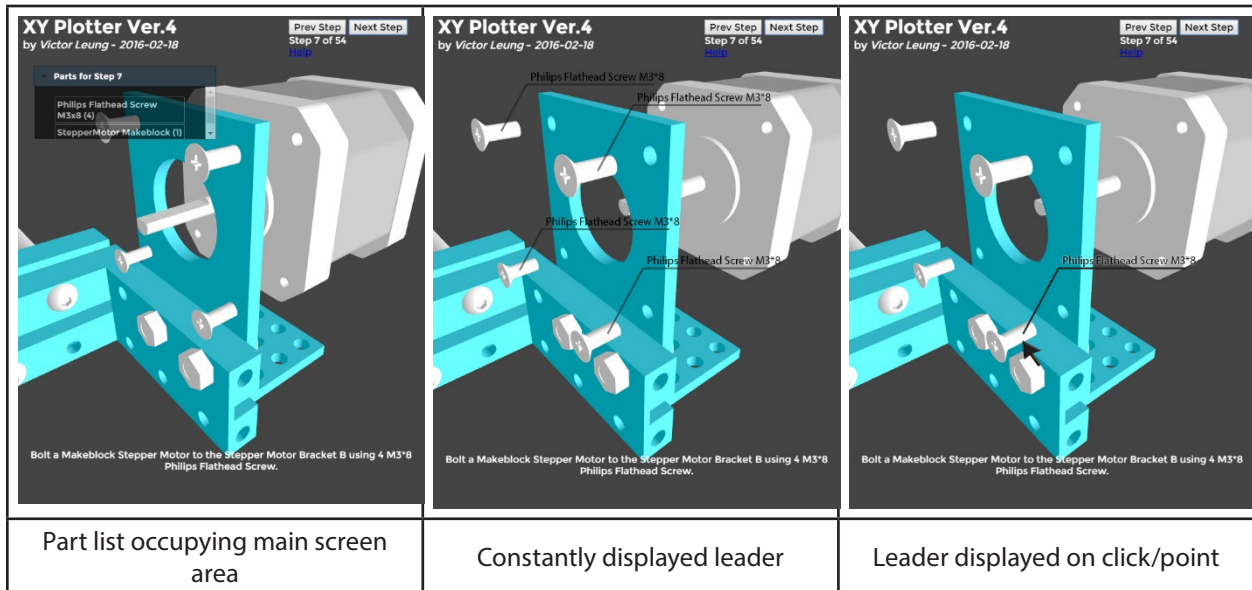
The third revision addresses a specific problem where different screw lengths are difficult to discern. The image below showed three different screw lengths (left to right: 12, 14, 16mm) which look very similar. The problem is worsened by the perspective projection of the camera. A part list with names is necessary to help users identify the right screw from the animation²⁵.

²⁵ Assuming students have screws that are properly labelled.



^ Three different screws look similar in perspective

The part list provides useful information, but it adds an extra panel on the screen. On devices with a small screen, the part list and the main 3D environment fight for screen space. This problem did not caused major trouble in my workshops because most students viewed the animation with a laptop. The image below is two other potential solutions to solve this problem, by showing the name tags as a 3D leader together with the 3D environment. The part list is (1) either constantly displayed, (2) displayed smartly only for confusing parts, or (3) displayed on user click.



5.3 Play / Pause mode

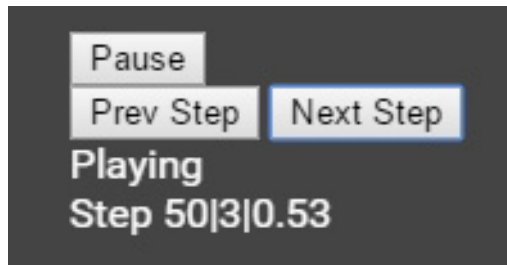
In the early implementation of the web player, there were three buttons for the user to control the playback: (1) Play/Pause, (2) Previous Step, (3) Next Step. Users were expected to click Play button once after the “Loading Stage”, to start the animated movements. Then users were expected to go forward or backward in step using (2)

and (3). This “Play/Pause” button was originally designed to allow users to pause at a specific moment during the animation cycle²⁶ for examining the objects.

²⁶ The animation cycle is typically 3s long followed by 1s pause before repeating itself.

However, during workshop #5, a team of two students did not realize the player is in “paused” mode when the Web Player started. They did not see the animated movement on the screen but they assumed it was normal. They used the player only by clicking button (2) and (3), which

still caused the player to progressively show the parts, but without the animated sequence. The team still managed to assemble the task but with substantially more difficulty, as reported in their feedback form.



^ The Pause Button



^ Removed the pause button and state

In the next iteration of the web player, the “Play/Pause” button was removed. The player is always in play mode, and the animation will always loop. This is a quick compromise to eliminate a potentially confusing feature, because there was no report from any students having trouble because they cannot pause the video. An alternative solution is to keep the button, but start the software in play mode.

Miller (2011) have explained the cognitive friction caused by multi-modal software state in his course in MIT. The Play/Pause button created two modes for the software and this caused a confusion when users entered the “paused” mode without realizing it. User interfaces are generally more novice-friendly when they do not have multiple modes.

This problem in fact points to more careful design of the player controls. The Web Player has many

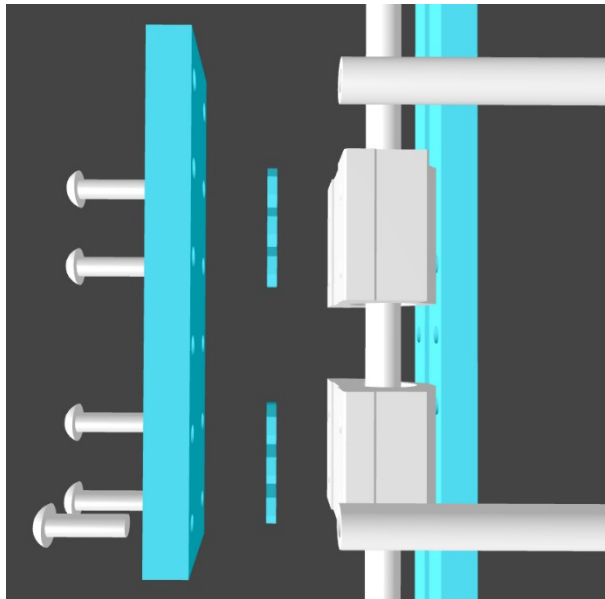
similarities to a video player (both load a file and play some video), and it is convenient to borrow UI elements from a video player that users are already familiar with. However, a simple task analysis can reveal that our web player receives much more user interaction (clicking button to advance step and camera control) than a traditional video player (where you load a file, click play and sit back). It is therefore worth rethinking the design of the player control. Below is a table of the relative frequency that a user access as the features, and can point towards future implementation of the interface.

Feature	Frequency
Forward to next step	Frequent
Back to previous step	Occasional
Seek/Scroll to previous or next steps	Never
Go to final step	Rare
See overall progress (see current step and total step number)	Frequent
Seek/Scroll animation within a step (change time / phase)	Rare
Pause animation (pause time / phase)	Rare
Toggle visibility of part list	Rare
Toggle visibility of caption	Rare

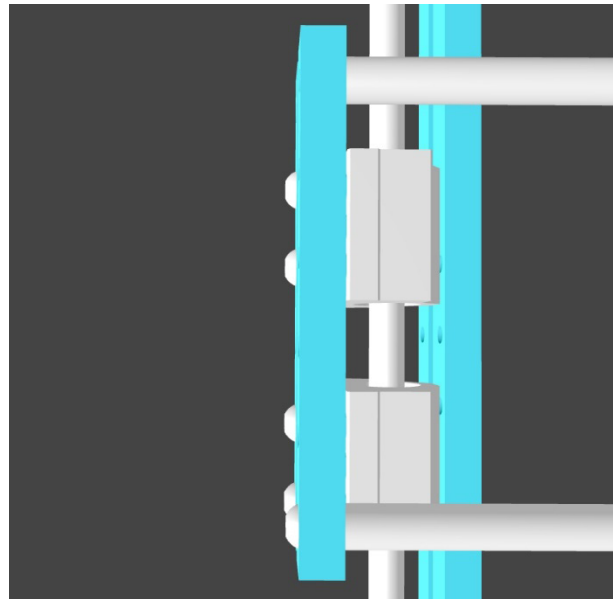
5.4 Occlusion

The animation itself did not provide any means for the user to see a complete list of all the parts in a step. Users need to pay attention to the parts that are animated on the screen to figure out

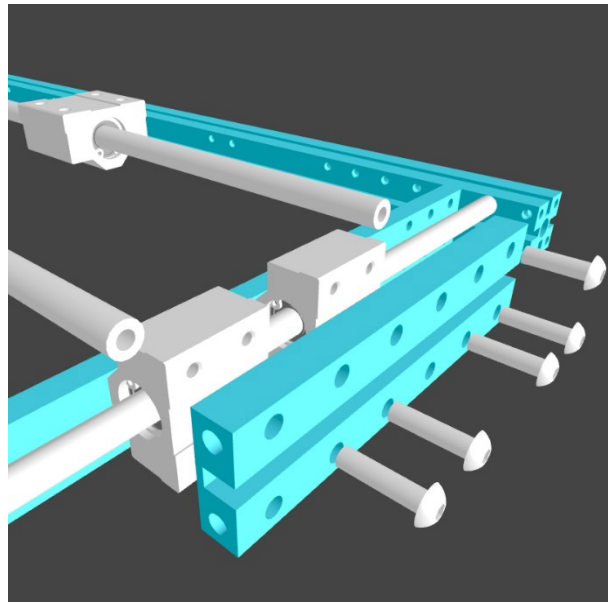
what is new in each step. With the ability to orbit the camera and choose any viewing angle, it is often possible for the users to accidentally pick an angle where some small parts are occluded by others, and users are not aware of it. For example there were two small shims in the plotter that were often left out by users.



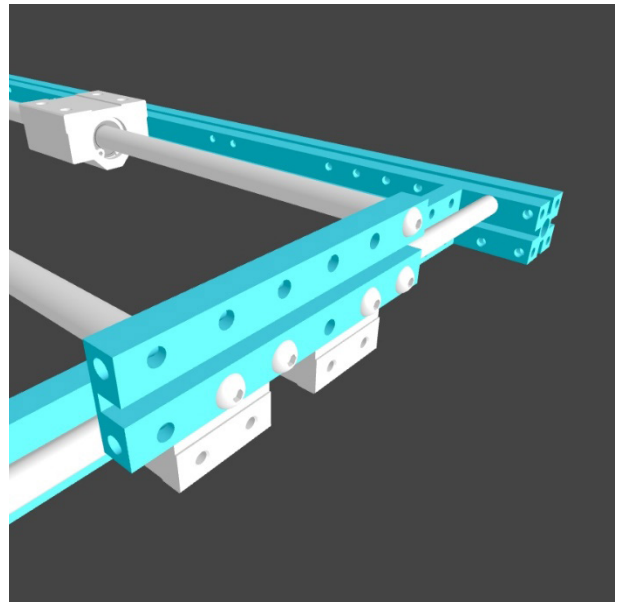
^ small piece **visible** from this angle



^ small piece **being occluded** when assembled



^ small piece **being occluded** from this angle



^ small piece **being occluded** when assembled

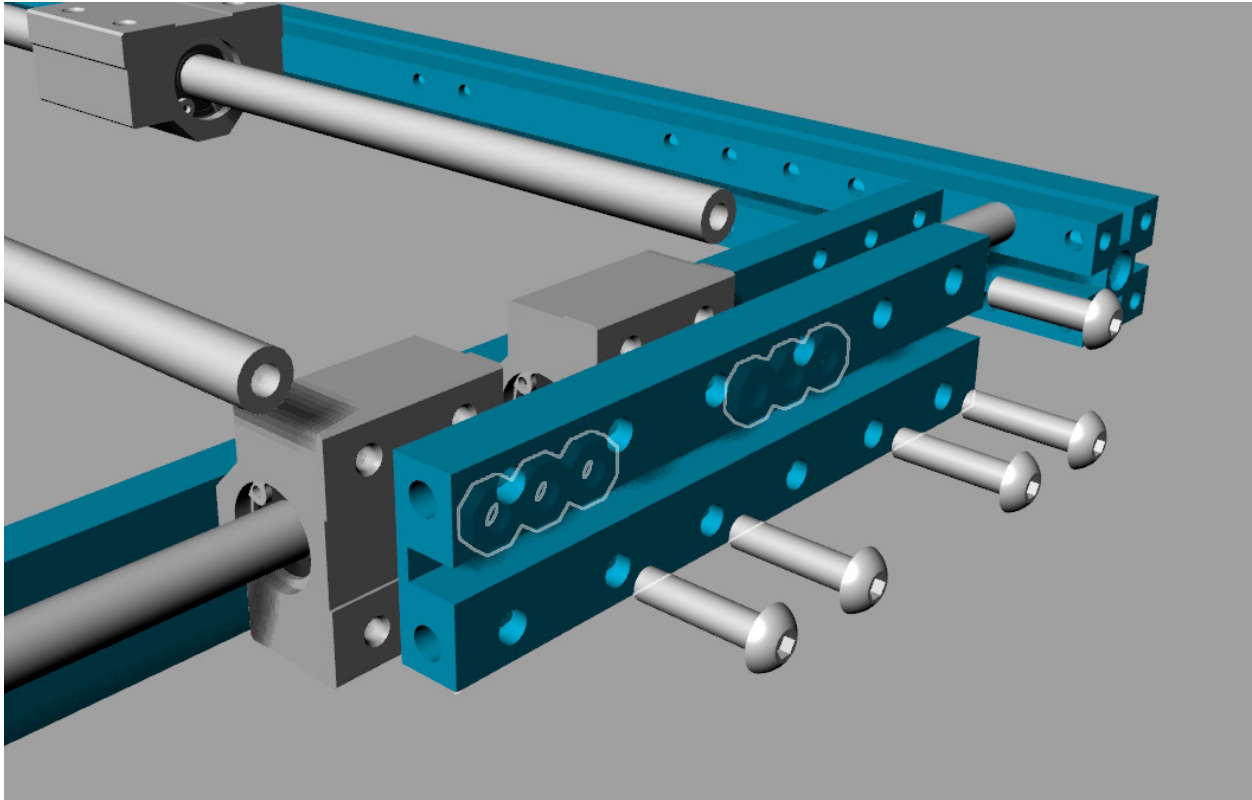
I once believed that the part list would provide awareness of all the parts in a step and can act as a counter-check for novices to avoid missing or forgetting parts. However, in practice, students did not consult the part list at the beginning of each step. Their main focus was the animated models and they only looked at the part list when they had doubt in identifying a part. This problem where users ignore information on a User Interface is discussed in the third chapter in Cooper (1999). He explained that users pay attention to information that they think is essential to achieve their goal and ignore the rest.²⁷

User interface can be designed to change user behaviour. For example, we can strongly enforce the part list in the beginning of each step by maximizing it, and require the user to click a button

to return to the animation screen. The maximised part list may even provide image cues of the necessary parts. This extra step encourages users to gather all the necessary parts from the material stack before watching the assembly animation, thus increasing the awareness of all the required parts, at the cost of intruding into more advanced user's workflow.

Another approach is to introduce occlusion visualization techniques to ensure users are aware of occluded parts. Kruger et al. (2006) described a number of Focus and Context (F&C) aware rendering techniques that combine occluded objects with the objects in front using a locally selective transparency to produce a see-through appearance. A simpler algorithm is shown below, to overlay a rendered silhouette of the occluded objects on top of the final image to hint their location. Users can then manually orient the camera to discover the occluded parts.

²⁷ What users think are important may not always be correct and thus they ignored important information.



^ Occluded silhouette overlay on rendered image

5.5 User interface navigation

Orbiting is the most frequently used camera interaction in the web viewer. This is because it is most convenient to orbit around an object of interest to understand the 3D geometry. However, in some cases when the user needs to shift from one object of interest to another, the user needs to pan the camera such that the invisible center of orbit lies close to the new object. I frequently found users who have no CAD experience having trouble performing this manoeuvre.

This manoeuvre is complex, first, the user needs to be aware of the imaginary center of orbit and move this center by panning. Second, the pan control of the camera is limited to directions

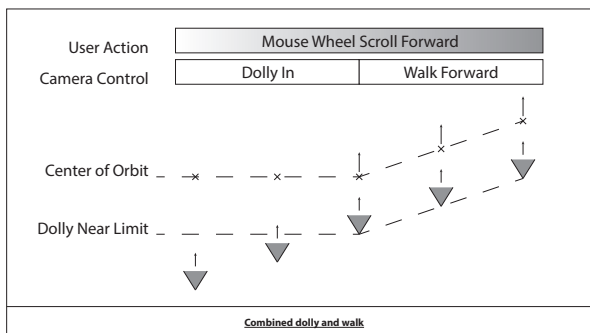
parallel to the screen (Pan left, right, up and down), thus it cannot pan forward or backward. Professional users often use the trick “pan , orbit 90°, pan, orbit 90°” to move the center of orbit to a desired location, but this is very hard for novices to learn. When novices attempt to use the closest matching function: to dolly in or out (which seems to move the camera normal to the screen), they will fail miserably because the dolly function does not actually move the center of orbit.²⁸ When the novices continue to dolly until their camera reaches the near or far limit of the dolly, they are puzzled why they cannot “zoom” anymore.

²⁸ Dolly only moves the camera towards the center of orbit.

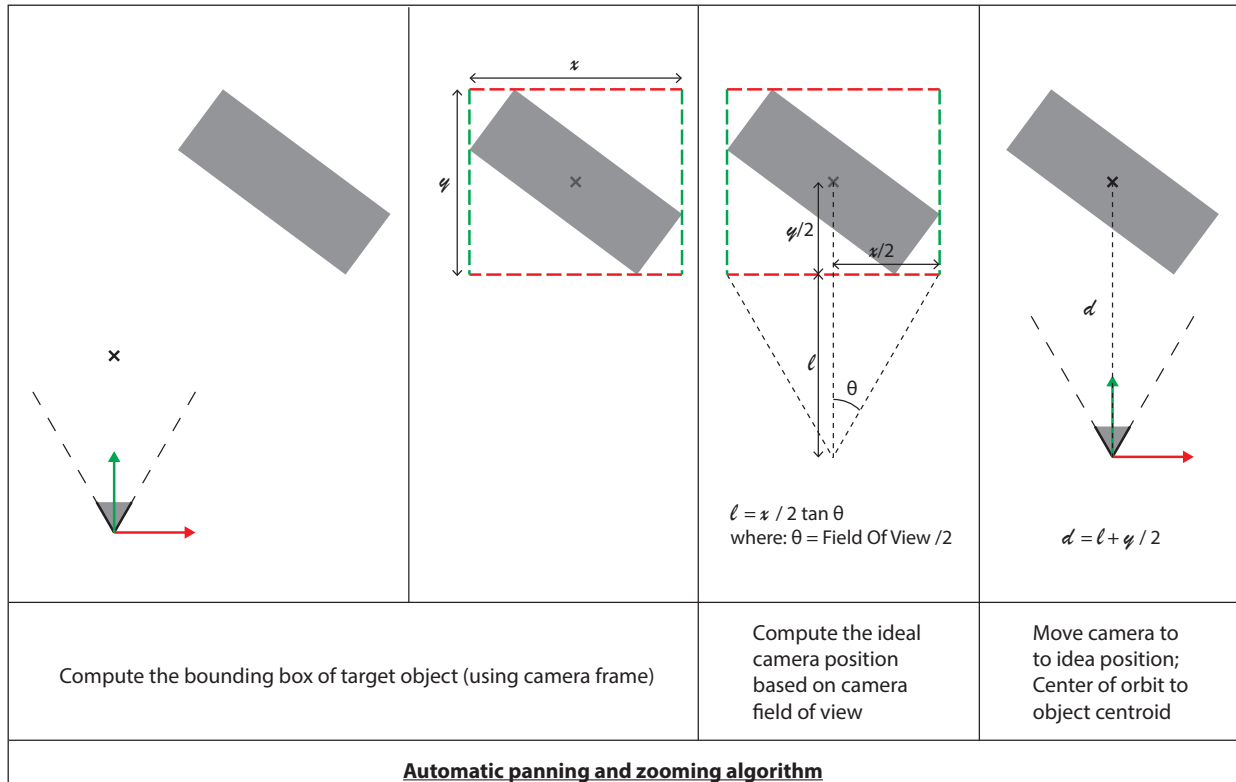
Camera Control	Dolly	Orbit	Pan	Walk
Front View				
Top View				
Center of Orbit Movement	Stationary	Stationary	Parallel to Screen	Normal to Screen
Different types of Camera Control				

control to view the different sides of the object, I experimented with an **automatic panning and zooming** feature that liberate the novice users from using panning at all. This feature automatically moves the orbiting center to the center of an object of interest. It also dollies the camera to a distance such that the object of interest is maximized on the screen but without clipping.

This whole confusion is inherited from the convention of professional CAD environment control, which is not natural for the novice to pick up. One potential suggestion is to change the mouse wheel function from dolly-in/out to walk-forward/backward, such that when the mouse wheel is scrolled, both the center of orbit and the camera position moves perpendicular to the screen. However, this will constrain the center of orbit and camera position at a fixed distance, and this means the user will not be able to orbit around a point at different distance. It is therefore best to bind the mouse wheel to a dolly function, and activate the walk function only when the dolly reaches its distance limit.



Observing how professional CAD users frequently pan the imaginary center of orbit such that it is in the center of an object of interest, and use orbit



In the 5th iteration of the Web Player, this feature is implemented and is triggered in the beginning of each step. Users still have the control to pan, if necessary. I have also animated the camera movement to give the user a direction cue of where the camera has moved. Note that this feature preserves the viewing direction of the original camera, which helps in preventing users from disoriented.

A variation of this feature is to move the camera and orbit target to an ideal, pre-determined location which has the best angle to show the assembly objects and motion. Agrawala (2003) provides some examples of how this angle can be computed automatically. Care should be taken such that users are not disoriented by the shift of camera orientation.

5.6 No mouse

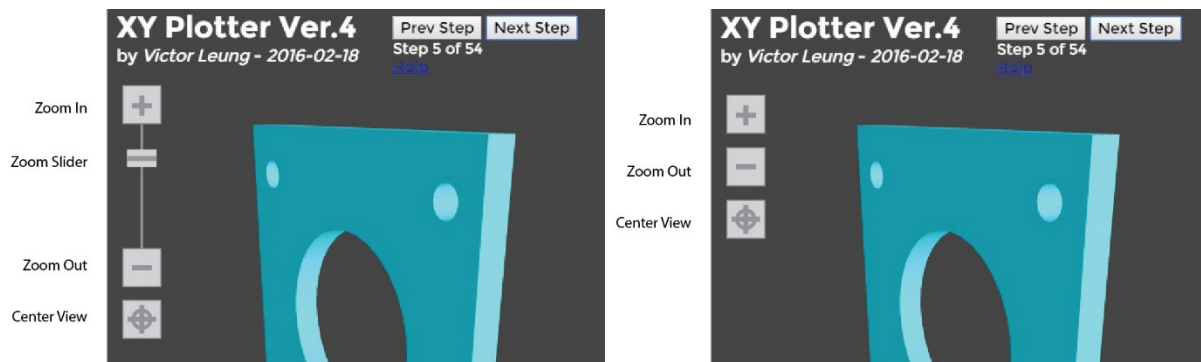
As laptop computers, tablets and smart phone become more popular, users of these portable devices frequently abandon the computer mouse, while professional CAD users still prefer a computer mouse for precise pointing. Because the Web Viewer can afford much less precise pointing, and it should cater to users without a computer mouse. The implementation of the camera control using OrbitControl.js in this thesis monitors both mouse and touch gesture.

	Mouse	Trackpad	TouchScreen
Pan	Hold and Drag Left Mouse	Hold Left Key and Drag on Trackpad	3 Finger Drag
Rotate	Hold and Drag Right Mouse	Hold Right Key and Drag on Trackpad	1 Finger Drag
Zoom	Mouse Wheel / Hold and Drag Mouse Wheel	2 Finger Zoom Gesture *	2 Finger Pinch

* Not supported by all computer trackpad driver

One problem I have observed is related to students using the trackpad on their laptop. Because most trackpads do not have a zooming wheel, it is not possible for them to zoom. A potential

solution is shown below. Buttons or a slider bar can be added on screen for zoom control, which the user can easily click or drag either by mouse or on touch screen.



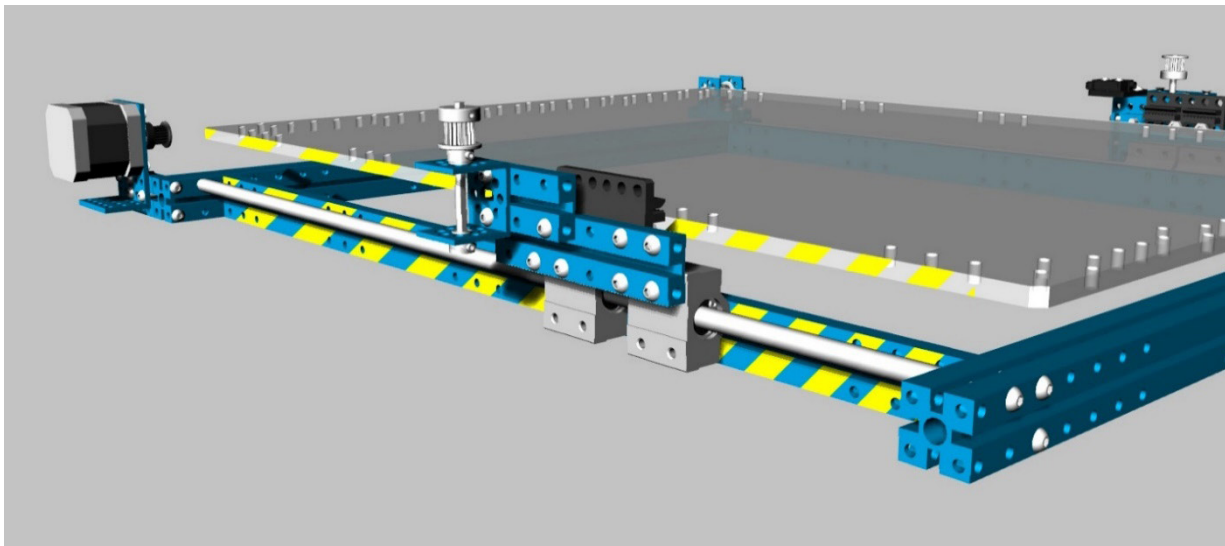
^ Two examples of zoom control implemented as UI elements

5.7 Showing alignment

During the assembly of a machine, it is often necessary to align edges or faces between different parts. In the plotter experiment, students were often not aware of alignment between parts. I realized that it is often enough to point out the alignment, by pointing to the pair of surfaces or edges on two parts. Therefore I suggest that fu-

ture implementation of the software incorporate a method to point out or highlight points, edges or surfaces to show alignment.

This alignment action can be implemented as a separate action in the graph, or embedded as a standard feature within all existing actions. Changes on the animation description file will also be necessary to accommodate this.



^ This image shows two faces being highlighted to draw attention.

5.8 Hard-to-animate steps

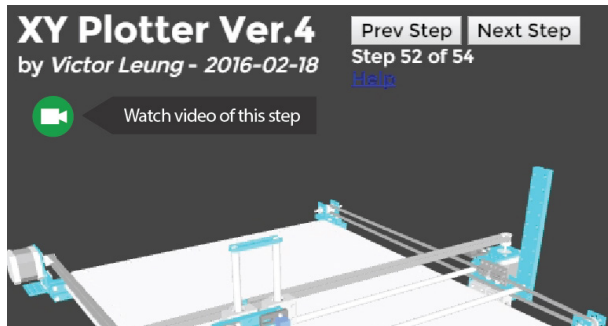
While mechanical parts are usually rigid, there are other applications where parts are flexible or deformable. When students were installing the rubber timing belt in the plotter, they need to bend the belt around multiple pulleys and threading it through holes and slots.

This type of motion is very difficult to encode using the keyframe interpolation method and

requires deformable mesh rigged with some skeletal control points. It is difficult to define such motions using the assembly graph, and also very difficult to animate using the interpolation methods we developed.

It is more practical to include a recorded video of the assembly process during these steps. This feature is not implemented in this thesis but a future version can easily incorporate this by adding a special action node to the CAD plugin to handle these steps and ask the designer for the URL of a

video. The Web Player should handle these nodes and display the parts using fade-in animation, and prompt the user to view the video.



5.9 Calibration and other non-assembling step

Precise machines often require some calibration. While the method of calibration varies substantially across different types of machines, we can analysis the general pattern of this assembly step. For example, in the workshop²⁹, the students need to calibrate the angle between the X axis and Y axis. They need to install a few **temporary parts** into the machine for alignment³⁰. The set screw on one of the pulleys is loosened and then the belt rotated until the temporary parts have no gaps on each side (**adjust and observe**). Then the set screw is tightened and the temporary parts are removed.

Temporary parts are not difficult to animate from the graphics point of view. However, the current

assembly graph structure cannot represent the actions to place a temporary part and then later remove it. It was designed to model how parts are connected. It is also difficult to show the “adjust” movement that transforms an uncalibrated assembly into a calibrated one, because those movements are often very small. A different visualization technique is required to inform the user where to look and what to look for proper alignment.

One solution is to learn from Ikea assembly instructions, which often include drawings of both the “Good Alignment” and the “Bad Alignment” state, with a big “Tick” or “Cross” to indicate which is correct. We need to introduce a new type of Action Node in the assembly graph to represent an Adjustment Action, because the assembled CAD model given to the CAD plugin is often in a perfect state of alignment. This Action Node can animate the parts to wiggle³¹ between two extremes of “Bad” state, to show the middle “Good” state. Future work will be required to study more examples of calibration actions and produce an effective visual communication method.

²⁹ In my workshop experiments, this calibration action is demonstrated by myself.

³⁰ In other cases, a measuring device is temporarily introduced.

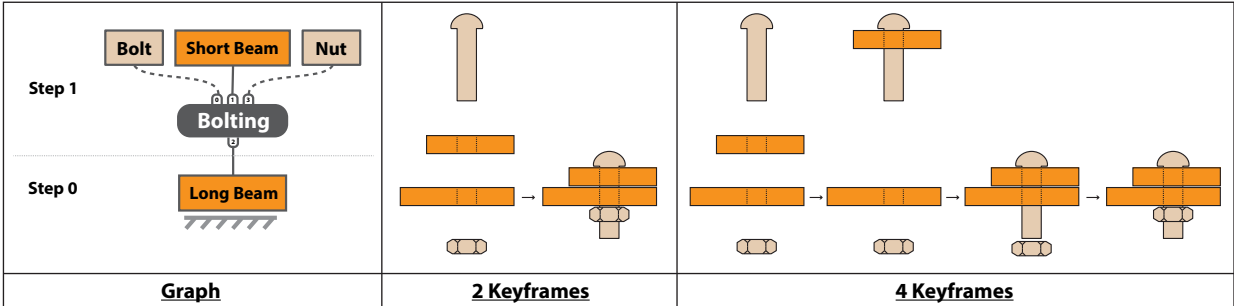
³¹ Wiggle motion can be a combination of translation and rotation. The designer needs to tell the software how that adjustment motion is made, so that the Adjustment node can compute two “Bad” extremes.

5.10 How do people actually assemble a bolt stack

The bolt stack animation strategy described in section 3.3 is based on a simple two-keyframe animation which works well for animating a bolt stack that contains only a few parts. However, for a more complex bolt stack, the motion of multiple parts coming together and (sometimes multiple) bolts being inserted simultaneously are far from the realistic limitation that human have only two hands and ten fingers. The absence of the human hand in all the animation requires the novice to infer where the hands and fingers have to be placed to hold the assembly.

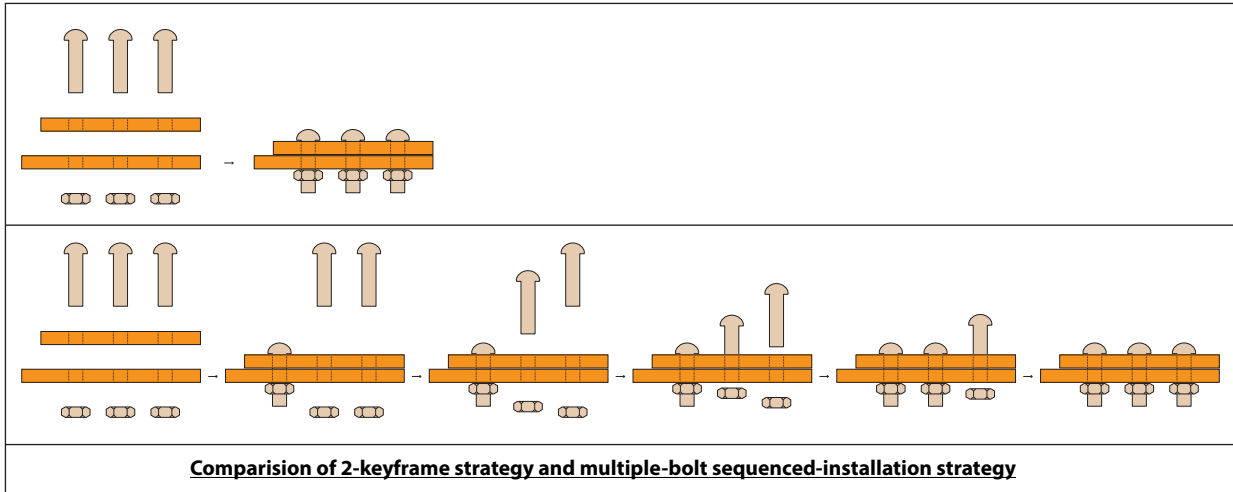
Students assembling a complex bolt stack often redesign the assembly sequence for each com-

ponent in the bolt stack based on what his hands can hold. It is more convenient for a human to assemble a bolt stack starting with components closer to the bolt head, because the bolt can act as a temporary alignment that holds the loose parts together. The example below shows a comparison between how the bolt stack was animated using the 2-keyframe strategy described in section 3.3, and a 4-keyframe strategy which resembles more closely to how a human assembles the stack. Note that the parts “immediately next to the bolt head”, are first inserted into the bolt, and that stack is then inserted into the static parts. This animation strategy is more descriptive but require more time to animate.



In the 2-keyframe strategy, when multiple bolts are grouped under one Bolting Action, they are animated together simultaneously. However, in reality, the bolts must be installed one after another; and sometimes, a particular order is required. It is therefore useful to animate the bolts one after another or with a small delay between them. This can be combined with the previous

strategy that animates the first bolt, and then subsequent bolts are animated one by one. The diagram below shows the keyframes of multiple bolts being animated with a delay between them. Note that the installation sequence can be easily understood.



In machine assembly, it is a common practice for professionals to install and hand-tighten all bolts in one step, before fully tightening all of them. This operation cannot solely be described using animation methods because hand-tightened bolts and fully tightened bolts look exactly the same. Future work can look into designing a visual representation to differentiate between hand-tightened and fully tightened bolt.

5.11 Sub-assembly

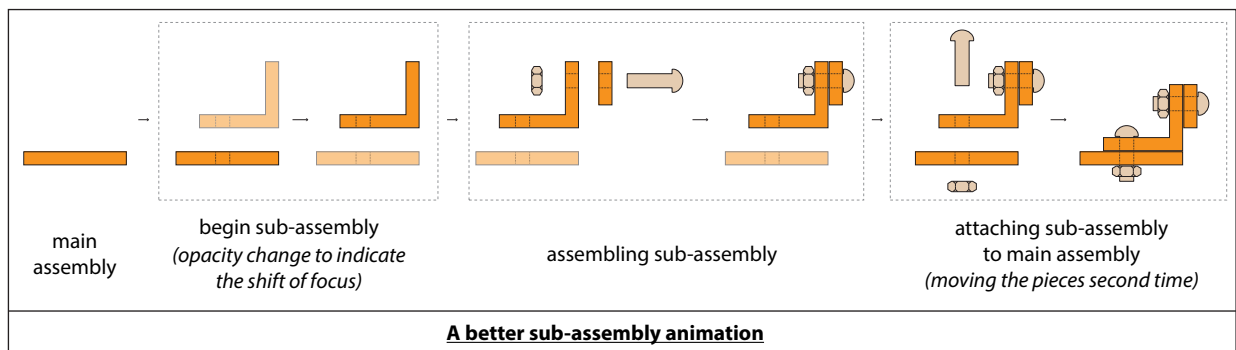
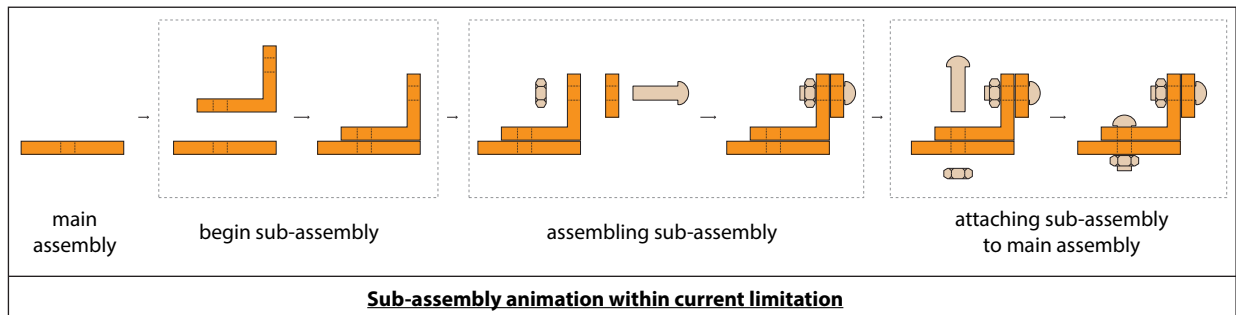
For complex assemblies, it is often beneficial to assemble a small sub-assembly and then later add it to the main assembly. A smaller sub-assembly at hand allows easier manoeuvring and reduces the effort to correct a mistake. It also contributes to a higher user satisfaction because of the final assembly goal appears more achievable when grouped into a few large steps. This also allows multiple people to work on the same assembly simultaneously by focusing on different sub-assemblies and then combining them.

One of the steps in the plotter animation shown in the workshops was a simple implementation of animating sub-assembly, using an Insert Action to substitute a Bolting Action. However, due to

the current limitation of the implemented Action Nodes, each Part Node can only be animated once³². The following diagram explains the difference between the sub-assembly I animated under this constraint, a normal linear assembly animation, and an ideal sub-assembly animation where parts needs to be moved twice.

32 Either one Insert Action, or one Bolting Action.

In the current implementation: One Action Node corresponds to the one animation of the Part Node. Although each Part Node can be connected to multiple Action Nodes, current implementation limits that, once the Part is inserted to the main assembly at a certain step, future connections to any Action Node, will not further animate the Part. The Part is considered static.



By animating the screw in the final step, I was trying to convey a sub-assembly operation, where some parts are assembled in a group, and the group is then assembled in the main assembly. However, limited by only one single movement, Part B was already in its final place during the first step of sub-assembly and that was confusing to most people. During the workshops, less than half of the students understood that animation was supposed to mean sub-assembly. They were instead wondering how to hold Part A and B temporarily while attaching Part C and D and later realized that they do not actually have to hold Part A and B.

One workaround of the single-move limitation is to hide the main assembly during the sub-assembly. However, care must be taken not to confuse the user as “Why did the assembly suddenly disappear?” A better solution is to support multiple-move, allowing the group of sub-assembly parts to be assembled adjacent to the main assembly, and then later attached to the main assembly. From the Web Player’s animation point of view, this can be easily implemented by introducing more keyframes in different steps. However, the user interface need to be carefully designed to explain “Put aside the main assembly”, and “We are now working on a sub-assembly”. The most challenging part is to create a proper representation of the sub-assembly with the assembly graph, for example, allowing the designer to group Part Nodes and Action Nodes together to form a sub-assembly. Future work should also investigate the properties of sub-assemblies in the graph, to allow automatic creation of this multi-step animation.

6.1 Baby steps

6.2 Avoiding mistakes

6.3 Beware of seemingly symmetrical parts

6.4 Warn novices about irreversible mistakes

6

Content preparation guidelines

This chapter contains practical guidelines for how to prepare the assembly graph. This advice is deduced from observations of student's performance during the workshops, as discussed in the previous chapter. They can be studied by the machine designer as application guidelines, or implemented in the CAD Plugin's code base to automatically check the assembly graph.

6.1 Baby steps

The mental cycle of a novice when viewing each instruction step in the Web Player goes like this:

(1) View the animation, understand the assembly action and put this action in memory, (2) Put all the part's description and quantity into memory. (3) Look for the part from the material pile. Recalling each part from memory. (4) Assemble the

parts, recalling the spatial orientation and movement from memory.

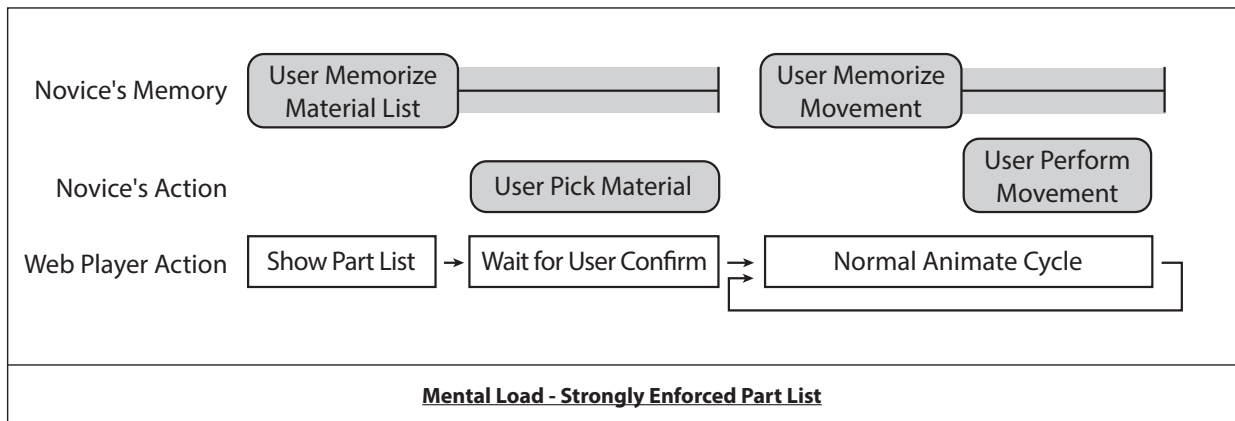
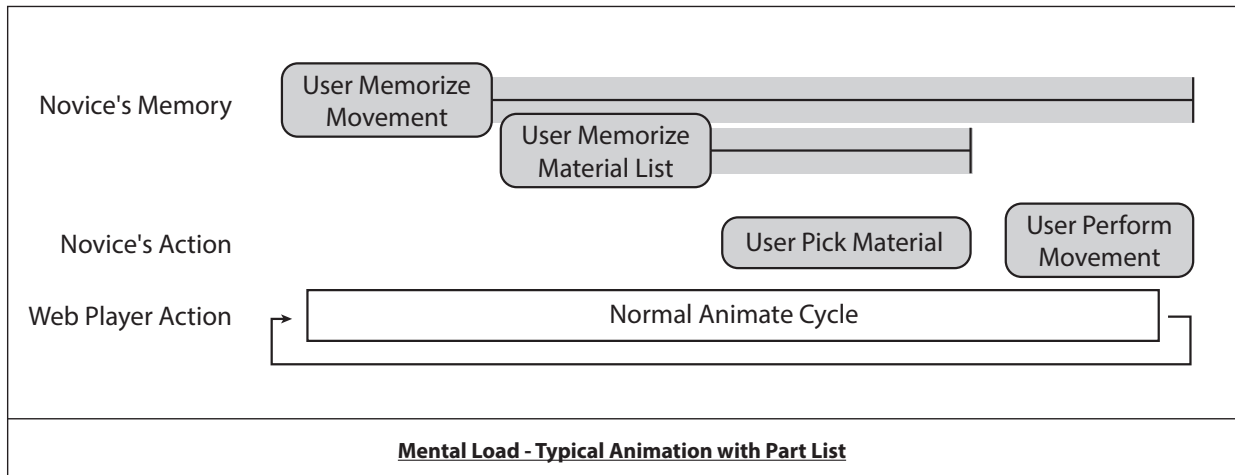
The problem is that human short term memory can hold only a few things for a short amount of time. It is generally believed that this short term memory or working memory can hold about 5-9 things, and last about 15 seconds (Miller 1956).

Also observed in the workshops, it is common for

students to search for the parts from the material pile, and then check the screen again to refresh their memory of the assembly movement again.

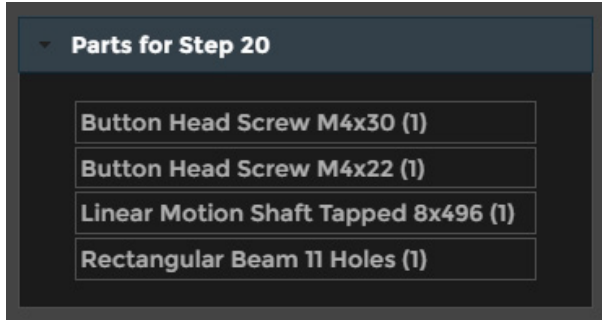
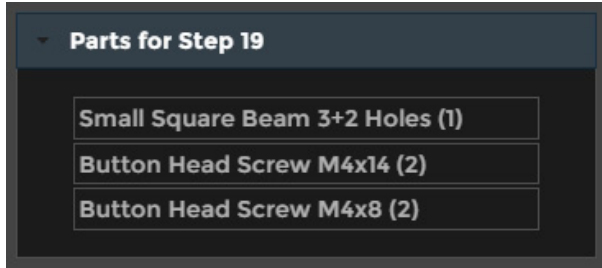
For the comfort of the novice, it is best to **separate an assembly into steps as small as possible**, to reduce the mental load and reduce error. As a general rule of thumb, do not contain more than 5 unique parts per assembly step. It is also possible for the Web Player to show the part list

and force the novice to pick the parts before watching the animation. The diagram below shows this strategy compared to the current method. Notice that if we show the part list first and force the user to pick materials before watching the animation, both amount and duration of short term memory is reduced. Yet, this method greatly reduces the enjoyment of the user because it substantially reduces “thinking”, and thus should be used only if enjoyment and fun is unimportant.



It is also very common for novices to mix screws of different length. In fact, any parts that look similar (for example M4x14 bolt and M4x16 bolt)

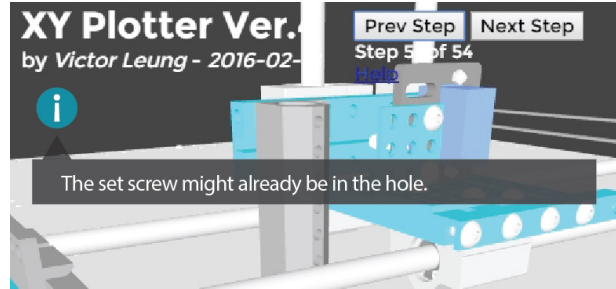
should not be placed in the same step. Below are two examples of similar looking screws being placed in one step.



6.2 Avoiding mistakes

When novices make a mistake and did not realize that immediately, it often takes a lot of time and effort disassembling parts to correct the mistake. Because the Web Player is designed to be a self-learning tool without any instructor's supervision, the users relies on visually comparing their ongoing assembly and the 3D model on screen to verify their correctness.

It is hard for the designer to estimate which steps are more prone to error, therefore a user study is generally the best method to reveal these potential problems. The designer should first attempt to fix these problems by redesigning the assembly sequence, otherwise an information text should be added to provide guidance. Below is an example of an information bubble.



6.3 Beware of seemingly symmetrical parts

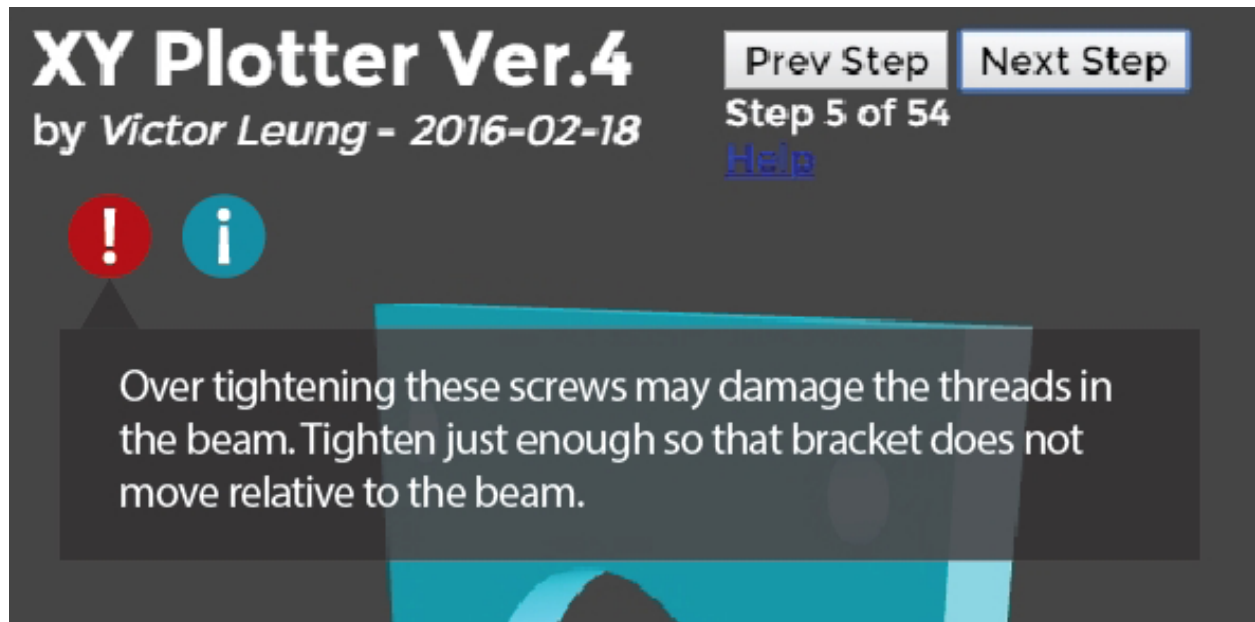
One of the most common mistakes students made is mis-orienting parts. Some parts may be very similar in their symmetry but must be installed in a specific orientation. Even if the 3D model in the animation has correctly modelled the small features of the asymmetry, users who have overlooked the part as a symmetrical part, will also tend to overlook the animated models.

Below is an example of an asymmetrical warning flag that is implemented in the definition of parts (in the part description file), to automatically raise a warning symbol next to the part list.



6.4 Warn novices about irreversible mistakes

If a mistake is not reversible, then it should be carefully communicated to the user via a warning message to warn the user of a potential damage.



^ Warning message displayed in the Web Player

7.1 New issues in instructions design

7.2 Allowing variations in the assembly step

7

Conclusion

The first part of this thesis proposed a new workflow to represent an assembly sequence and assembly motion using a graph structure. This graph can be easily created and edited by designers and then interpreted by computers. The software can then automatically create 3D animations that can be presented to novices using a Web Player. I named this workflow *Put It Together*. This instruction method has the following advantages over the traditional methods such as paper-based instructions, sharing static CAD models or watching demonstration videos.

- Novices can see the animated motion of the parts from any angle.
- Novices can follow the animation and progress at their own pace.
- Designers can create and edit the animation easily.

The second part of the thesis demonstrated the Web Player being used in an educational environment, showing mechanical assembly instructions to students. Discrepancies between expected and observed performance were recorded and analysed. New issues that arose in such settings were documented, and suggestions for future improvements were made. The majority of students considered the tool an improvement over traditional methods primarily due to its effective communication of movements.

7.1 New issues in instructions design

While the primary goal of the animation is to allow the unambiguous communication of the assembly process, we should be careful **not to treat novices as robots** executing instructions. Not only does this undermine the ability of the human brain to perform tasks that are far superior to what the animation can describe, it also takes away the learning experience by which novices process information with their brains and make new connections from their past experiences.

The Web Player is designed with utmost flexibility, allowing users to customize the user interface to suit their levels of competency. Helper features should be able to provide more information to novice users while not causing disturbances to more advanced users. This flexibility is not possible from previous instruction methods, such as illustrations or videos. The features implemented in this thesis encompass this design approach, which can be turned off easily by the user.

Communication between experts and novices is always a difficult task. Experts care about precision and efficiency, speak in technical jargon, and make incorrect assumptions. Novices vary in experience, skills and intentions. They make mistakes, don't understand technical limitations, and don't like to follow instructions. It is not easy to create a communication conduit between experts and novices while respecting both of their goals.

The *Put It Together* tool chain offers a new perspective in thinking about instruction design.

This is a result of understanding designers' CAD routines and their struggles with preparing traditional instructions. The clever use of the assembly graph allows designers to easily express their intent and focus on designing a good assembly sequence. This software does the hard work in converting this into an animation and presenting it with an effortless interface so that novices can focus on following the instructions.

7.2 Allowing variations in the assembly step

The aim of this thesis is to communicate a set of pre-defined steps that lead to one pre-determined outcome. The process does not demand nor suggest much creativity. The creative design process of the assembly ends as the designer finishes the assembly graph and exports the files.

However, this thesis still fits into a creative learning scenario. In the case where the goal is to develop creativity in students, the animated instructions in this thesis fit into the first stage of learning where students learn by mimicking. El-Zanfaly (2015) provides a context for how mimicking can fit into a complete learning model. For example, an educational website can utilize the animated instructions to guide students through various assembly exercises within a larger curriculum.

In the case where the users need to perform an assembly task only once³³, or the speed and accu-

racy of the assembled object is prioritized³⁴, the Web Player can be developed to include various features to enforce a very simplistic “do as I say” instruction. It is effective in reducing misunderstanding and avoiding mistakes.

34 For example a mechanic performing a one-time repair to an unfamiliar car.

8.1 Screwdriver and the hand

8.2 Rethinking the assembly graph for temporary objects

8.3 Animating glue, paint, tape and other nonrigid bodies

8.4 Integration with robotic sets

8.5 Integration with discussion and feedback

8.6 Embeddable Web Player

8.7 Quantifiable results through web deployment

8.8 Making grammars in robotic sets

8.9 Assembling electronics, furniture, architecture and other things

8.10 Decoupling content and presentation

8

Future Work

8.1 Screwdriver and the hand

As discussed in section 5.10, the presence of a human hand or the installation tool in the animation can help novices better understand where to hold the parts and how certain motions are performed. It will be helpful in certain difficult situations such as: (1) Assembly that requires many different tools. (2) Assembly that requires tool change frequently. (3) Assembly that requires positioning the tool or hand in an unobvious position or moving it along an unobvious path.

The current implementation of the tool chain supports the use of recorded video clips to show complex motion or complex hand coordination. Further work is required to study an effective ani-

mation strategy for tools and hand and to validate its effectiveness in the mentioned scenarios.

8.2 Rethinking the assembly graph for temporary objects

As discussed in section 5.8 and 5.9, it is common in mechanical assembly to use temporary guides and props to assist assembly and alignment. It is also useful to be able to show removal of packaging materials in the animation.

The current set up of the assembly graph is a representation of the connectivity relationship between parts, thus not accommodating temporary objects and their removal from the scene after they are installed. Further work needs to be done to rethink assembly graphs to accommodate

temporary objects. This is challenging because temporary objects are often not modelled in the designer's static CAD model. These temporary objects may collide with a permanent object if they are modelled in the same space. Future work should seek a solution that can represent the relationship while avoid changing the established CAD workflow for the designers.

8.3 Animating glue, paint, tape and other nonrigid bodies

There are other types of non-rigid body transformations that are worth exploring. Non-rigid bodies such as belts and cables are common in machines; press fit fasteners, tapes and rope knots are common in consumer products and furniture. Animating these non-rigid bodies typically require deformable mesh models rigged with a skeleton. Thus is difficult for designers to model and animate. Deformable mesh models are also typically not supported in CAD software. Research into an easier modelling interface for the designer will be the first step in allowing non-rigid body animation.

Other types of actions, such as applying glue and paint, are hard to visualize because their 3D models change volume across time. Existing animation techniques for animating liquid uses a simulation based technique to generate a 3D model of the liquid frame by frame. However, this is difficult to model and slow to compute. Future work should study how liquid materials are applied in an assembly process, and how that can be abstracted and represented in the assembly graph and the animation description file.

8.4 Integration with robotic sets

It is worth noting the close relationship between instruction design with many kit-of-parts learning sets offered in the market (such as Makeblock, Vex Robotics and Lego). Most of these building systems are open-ended systems where users can explore different combinations. It is common for these companies to produce paper-based instructions for their users to learn the basics of the system before creating on their own.

The *Put It Together* tool chain can be used as a drop-in replacement for paper-based instructions, offering the users a better experience and reducing the effort for the designers to create them. Most of these robotic kits have a finite set of parts in their catalogue, which can be converted into a CAD library as described in Section 3.1.

Users who are capable of creating CAD models can use the CAD Plugin to create their own instructions and contribute their own designs. Online instruction sharing platforms such as Instructables.com have already proved popularity among hobbyists to share their knowledge. The *Put It Together* Web Player can be easily integrated with an inline frame on HTML web pages on these platforms to show animated instructions.

8.5 Integration with discussion and feedback

The Web Player can also be integrated with online discussion boards. Compared to other online

instructions that incorporate a discussion board after the main content, *Put It Together* discussions can happen separately at every step of the assembly process, offering highly specific discussion for users to talk about a particular step. These discussions provide valuable feedback to the designers to review and improve the instructions.

8.6 Embeddable Web Player

In order to broaden the application of the Web Player, it is best to develop an embeddable player that can be inserted into other web pages. For example, an education web site may include an assembly animation of only a few steps to illustrate a point, or a furniture company can show the assembly sequence next to their product information.

8.7 Quantifiable results through web deployment

Online software can easily collect usage information in real time. The *Put It Together* Web Player has the potential to offer a large amount of **user studies** to research on User Interface design and Instruction Design. Previous methods have largely relied on psychological analysis due to the high cost involved in user studies.

Future work should determine which type of user interaction can be quantified and what potential conclusions can be drawn. For example, time spent on each step can provide insight in **measuring complexity** and **improving time estimation methods** for mental and motor tasks.

Angles from which users view the 3D model can prove assumptions about best viewing angles and improve camera control algorithms. For example, Blanz et al. (1999) have empirically shown that people have a strong preference for viewing most objects from above and at oblique angles, rather than front or side views.

Experiments can be set up to vary UI elements, colour, features, and controls to gather quantifiable data about effective communication.

8.8 Making grammars in robotic sets

When *Put It Together* is used for sharing robotic designs for robotic systems such as Makeblock, Vex Robotics or Lego, it is possible to design computer algorithms to analyse the user contributed designs and understand how expert designers assemble parts. Future work can explore the automatic creation of assembly rules for creating an assembly grammar. Formalized knowledge in the form of making grammar will allow novices to create novel machines. For example, (1) analysis of how linear and rotary stage can be combined will provide a useful set of case studies for learners. (2) Analysis of different ways beams are joined together can provide a set of geometrical and functional grammars for joints.

Such work may also contribute to develop artificial intelligence systems that can design novel machines automatically or provide assistance for human designers in an interactive manner. For example, Umetani et al. (2014) showed a data-driven

method to create an algorithm that can assist human designers in designing paper airplanes.

8.9 Assembling electronics, furniture, architecture and other things

The *Put It Together* tool chain can be applied to many other types of mechanical assembly. As long as the assembly process and part relationships fit within the limitations discussed before, it can be converted easily into an animated animation. For example, Ikea flat-pack furniture often requires end-users' assembly. For instruction sheets to be meaningful across language and culture, they are often carefully designed with illustrations and use pictograms instead of text. Yet, it is still common to hear users' complaints. The *Put It Together* animation directly shows the assembly action, reducing the dependency on verbal explanation and avoiding miscommunication. It can be used as a drop-in replacement or supplement to their current illustrations.

Electronics is another common object often assembled by hobbyist. While digital files of circuit diagrams and PCB drawings can be shared easily online and personal milling machines are highly automated for milling PCBs, the tasks of assembling and soldering circuit boards are still highly manual. Further research should study the assembly process of electronics, such as cable wiring, crimping, part insertion, lead trimming and soldering.

In circuit board assembly, it is worth looking at the work done in automatic pick and place ma-

chines and how they are programmed. It is also interesting to develop CAD plugins for PCB design software.

8.10 Decoupling content and presentation

The current workflow where: (1) the CAD Plugin compiles the assembly graph into an animation description file, (2) the user shares that file online, (3) the Web Player interprets the JSON file, is a remnant of the workflow of creating computer animation. The CAD Plugin is an analog of animation software such as Autodesk Maya. The JSON file is an analog of a compiled video file such as .mov file, which can be uploaded and shared.

Future work should look into the elimination of the assembly description JSON file entirely, so that the compilation step can be avoided and the description file is no longer needed. The Web Player will interpret the assembly graph directly to animate the CAD models. Although this method requires more computational power from the viewer's client browser, it has many other advantages:

- The Web Player can read all the information from the assembly graph and decides how to animate the assembly. This provides more flexibility to Web Player developers to design the player without worrying about the pre-compilation leaving out certain information.
- It potentially allows the Web Player to customize animation style, speed and keyframes,

to accommodate user's competency and the device's hardware computing power.

- Less software maintenance is needed when developing many plugins for different CAD software. There will be no precomputation algorithms to maintain.
- The file being exchanged on the Internet is the Assembly Graph. This systematic setup encourages people to share the full source code of the assembly. (The Assembly Graph is similar to source code.)

9.1 Electronics used for the workshops

9.2 Software used for the workshops

9.3 Workshops Curriculum (6 hours x 5 days)

9.4 Workshop Curriculum (4 hours x 2 days)

9.5 Sample Workshop Questionnaire (used on 2016-02-27)

9.6 Animation Description JSON Schema

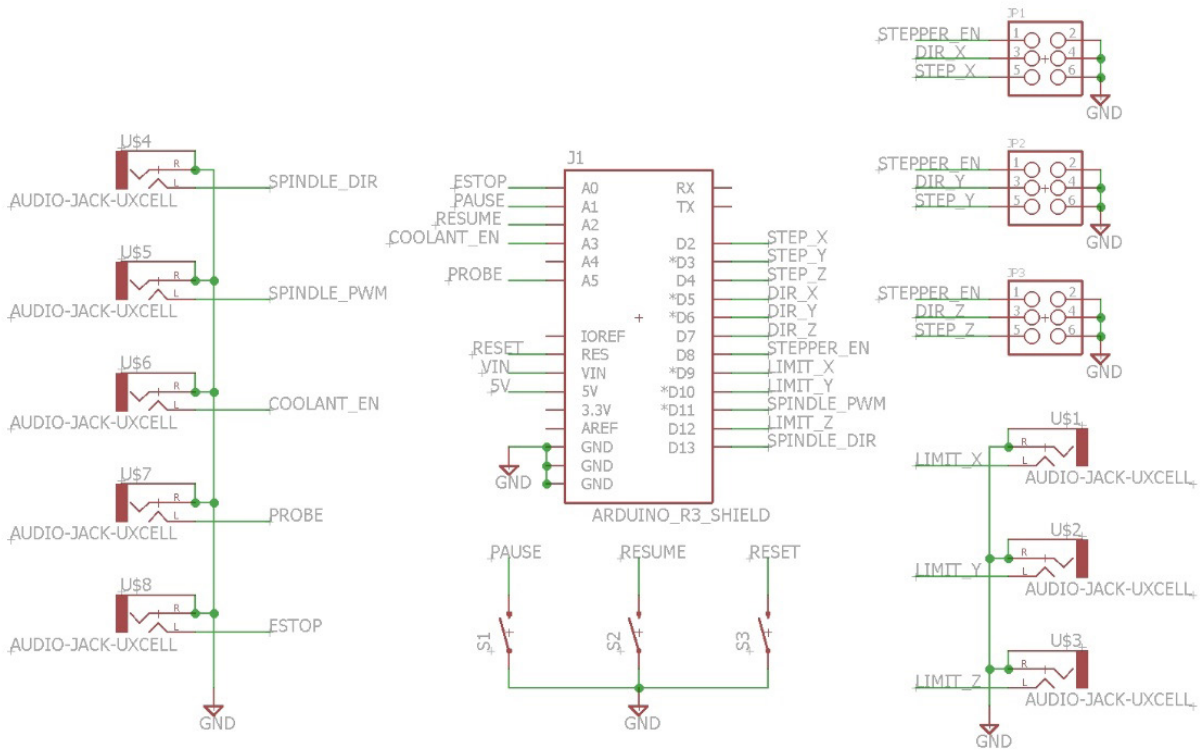
9

Appendix

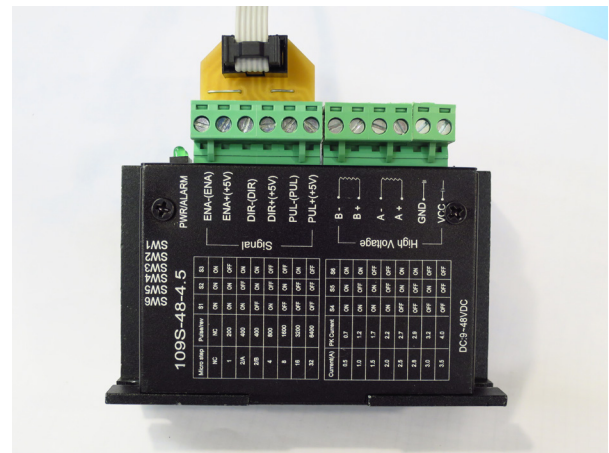
9.1 Electronics used for the workshops

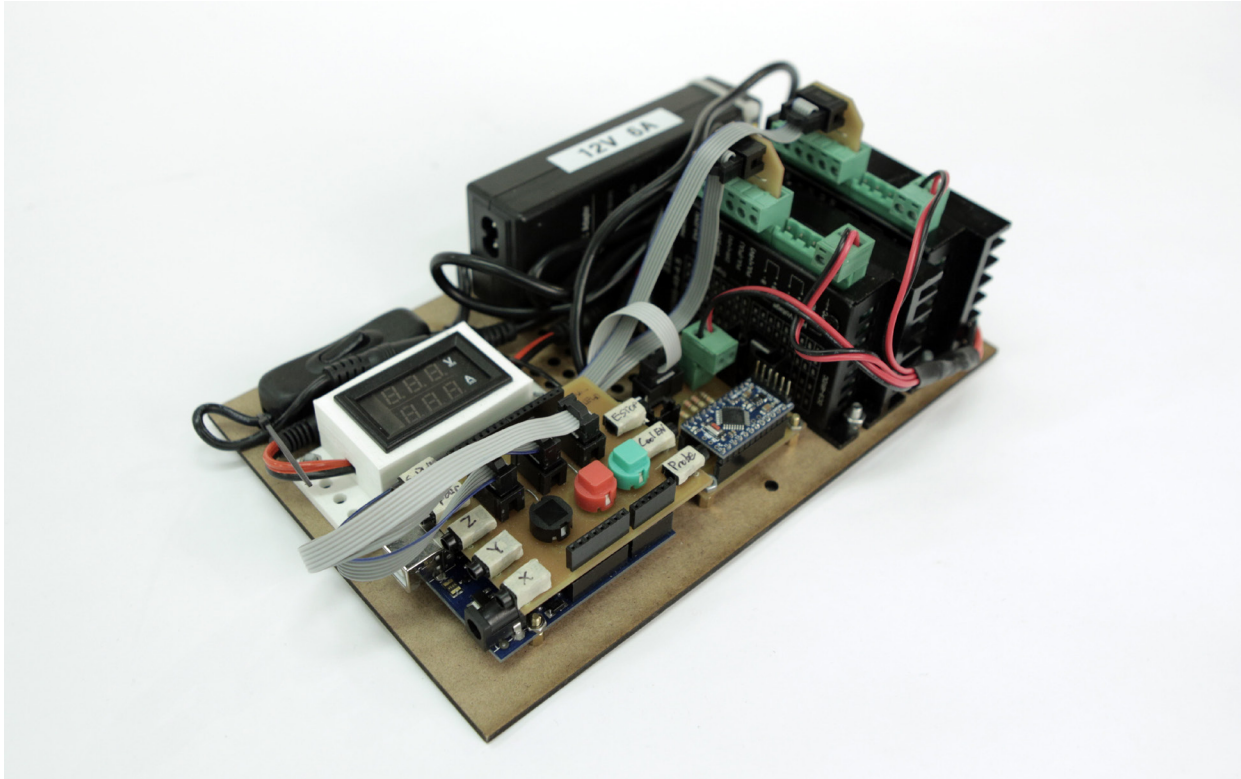
The controller of the plotter is Arduino Uno with ATmega328p microcontroller. Mitchell Gu had

created an adapter board (shield) for relaying the pins to various sockets. These sockets allows students to securely connect motor drivers and limit switches very quickly without tools. The schematic of the shield is shown below.



The stepper motors are driven by industrial stepper driver running on TB67S109AFTG stepper driver IC, it has reverse power polarity protection and fool-proof sockets. It can drive motors up to 3.5A, up to 32 microstepping. During the workshop, it was set at 2.0A and 8 microstepping. I designed a breakout board to receive data connection from a 6-pin ribbon cable via a shielded header socket.





9.2 Software used for the workshops

CAD CAM software

Adobe Illustrator CS5 / CS6 / CC was suggested to students as a tool for creating 2D vector drawings. "Live Trace" was demonstrated to students for converting raster images to vector drawings.

Rhinoceros 5.0 was used for creating 2D vector drawings. Students are taught how to create and edit 2D lines, arcs and splines.

Grasshopper 1.0 was used as a scripting platform for Rhinoceros 5.0. I prepared a script for the students to convert 2D drawings to G-Code. It

allows students to specify drawing sequence and drawing speed.

Controller software

grbl v0.9j was loaded on Arduino Uno for controlling synchronized motion of multiple motors. Universal G-Code Sender v1.0.9 was used from student's laptop, to stream G-Code files to grbl through USB connection.

9.3 Workshops Curriculum (6 hours x 5 days)

Day	Content	Duration (min)
Day 1	Theory: Stepper / Control	30
	Exercise: Run a stepper motor	15
	Theory: Parts Terminology / Gear Ratio / Microstepping / Belt Pulley	30
	Exercise: Basic Makeblock Use / Linear Stage	60
	Discuss: Guide / Drive / Precision / Stiffness / Homing Cycle	60
	Theory: Grbl settings / G-Code	60
	Exercise: Homing the linear stage	30
	Exercise: Design a 1 axis application	30
Day 2	Theory: Types of CNC machines / End Effector	60
	Exercise: Assemble and test XY Plotter	240
Day 3	Guest Lecture: Background of drawing machines / pen plotters	90
	Theory: Programming G-Code / Scripted Drawing	30
	Exercise: Make drawings / Brainstorm about what drawing method / media	210
Day 4 / 5	Modify the machine or code some special drawings	240 ea.

9.4 Workshop Curriculum (4 hours x 2 days)

Day	Content	Duration (min)
Day 1	Introduction to CNC technology, motor control and motion transmission	30
	Introduction to Makeblock modular robotic system, fasteners and tools	10
	Assemble XY Plotter	240
Day 2	Calibrate and test the plotter	20
	Introduction to G-Code programming and drawing conversions	30
	Program and make drawings with the plotter	210

9.5 Sample Workshop Questionnaire (used on 2016-02-27)

What is your Department / Course / School? _____

Check what you previously know

<input type="checkbox"/>	use a Screwdriver	<input type="checkbox"/>	program an Arduino (C++)
<input type="checkbox"/>	use a Hex Key	<input type="checkbox"/>	use a Solder Iron
<input type="checkbox"/>	use a Laser Cutter	<input type="checkbox"/>	designed and made PCB
<input type="checkbox"/>	use a Water Jet Cutter	<input type="checkbox"/>	program in Rhino Grasshopper
<input type="checkbox"/>	program a CNC Mill	<input type="checkbox"/>	use Autodesk Inventor
<input type="checkbox"/>	program a CNC Lathe	<input type="checkbox"/>	use Rhinoceros
<input type="checkbox"/>	program a Robotic Arm	<input type="checkbox"/>	use Solidworks
<input type="checkbox"/>	taken "How to make"	<input type="checkbox"/>	draw illustration
<input type="checkbox"/>	wired a Stepper Motors	<input type="checkbox"/>	paint painting
<input type="checkbox"/>	wired a Hobby Servo Motors	<input type="checkbox"/>	used Pen Plotter

There must be some difficulties you experienced during the assembly of the XY plotter. What instructions could have made that easier?

What are the instructions or information that you think is important but wasn't communicated through the animation?

How does the step-by-step animation help you

	-1	0	1	2	3	4
	Made things worse	Made no difference		Somewhat helpful		Very Helpful
Helped me understand the assembly sequence						
Helped me identify parts						
Helped me assemble faster						
Helped me avoid mistakes						
Helped me understand the functions of the parts						
Helped me understand how parts fit together						

The first half of the animation have captions, the second half does not. Do you think the presence of the captions helped you assemble? If yes, in what way did it help? If no, do you think I can remove it?

Did you find the yellow border highlight helpful?
(Highlights parts in current step)

The first half of the animation have captions, the second half does not. Do you think the presence of the captions helped you assemble? If yes, in what way did it help? If no, do you think I can remove it?

What could have been improved in today's curriculum? (To make it more satisfying for you) Or just anything else you want to say.

9.6 Animation Description JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "AnimationDescription",
  "description": "An animation generated by Put It Together toolchain",
  "properties": {
    "animationTitle": {
      "type": "string"
    },
    "animationAuthor": {
      "type": "string"
    },
    "animationDate": {
      "type": "string"
    },
    "blocks": {
      "description": "Block Instances used by the animation",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "blockName": {
            "description": "The unique identifier for a block",
            "type": "string"
          },
          "objFileName": {
            "type": "string"
          },
          "mtlFileName": {
            "type": "string"
          },
          "mtlExist": {
            "type": "boolean"
          },
          "metadata": {
            "description": "Additional metadata in Key-Value Pair",
            "type": "array",
            "items": {
              "description": "the value",
              "type": "object",
              "properties": {
                "key": {
                  "type": "string",
                  "required": true
                },
                "value": {
                  "type": "string",
                  "required": true
                }
              }
            }
          }
        }
      }
    },
    "required": [
      "guid",
      "blockName",
      "objFileName",

```

```

        "mtlExist"
    ]
}
},
"objects": {
    "description": "Represent each instance of instantiated block",
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "guid": {
                "description": "The unique identifier for this instance",
                "type": "string"
            },
            "blockName": {
                "description": "The identifier pointing to a block",
                "type": "string"
            },
            "objectName": {
                "type": "string"
            }
        },
        "required": [
            "guid",
            "blockName",
            "objectName"
        ]
    }
},
"steps": {
    "description": "Represent each animated step",
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "id": {
                "description": "Unique chronological number of this step",
                "type": "integer"
            },
            "caption": {
                "description": "Textural instruction of this step",
                "type": "string"
            },
            "objects": {
                "description": "List of animated objects in this step",
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "objectID": {
                            "description": "The identifier pointing to an object",
                            "type": "string"
                        },
                        "startMatrix": {
                            "description": "4x4 affine transform matrix in starting position",
                            "type": "array",
                            "items": {
                                "type": "number"
                            }
                        }
                    }
                }
            }
        }
    }
},

```



```

        "endMatrix": {
          "description": "4x4 affine transform matrix in ending position",
          "type": "array",
          "items": {
            "type": "number"
          }
        },
        "required": [
          "objectID",
          "startMatrix",
          "endMatrix"
        ]
      }
    },
    "required": [
      "id",
      "objects"
    ]
  }
},
"required": [
  "animationTitle",
  "animationAuthor",
  "animationDate",
  "blocks",
  "objects",
  "steps"
]
}

```


10

Reference

- Agrawala, Maneesh and Phan, Doantam and Heiser, Julie and Haymaker, John and Klingner, Jeff and Hanrahan, Pat and Tversky, Barbara. 2003. "Designing Effective Step-By-Step Assembly Instructions." SIGGRAPH 828-837.
- Card, Drew, and Jason L. Mitchell. 2002. Non-Photorealistic Rendering with Pixel and Vertex Shaders. Wordware Publishing, Inc.
- Cooper, Alan. 1999. The inmates are running the asylum. Indianapolis, IN: Sams.
- El-Zanfaly, Dina. 2015. "[13] Imitation, Iteration and Improvisation: Embodied interaction in making and learning." Special Issue: Computational Making, Design Studies 79-109.
- Erik, Dam B. and Martin, Koch and Martin, Lillholm. 1998. Quaternions, Interpolation and Animation. Technical Report, Denmark: Department of Computer Science, University of Copenhagen.
- Gooch, Amy, Bruce Gooch, Peter Shirley, and Elaine Cohen. 1998. "A Non-Photorealistic Lighting Model For Automatic Technical Illustration." Proceedings of SIGGRAPH 98 447-452.
- Gooch, Bruce, and Amy Gooch. 2001. Non-photorealistic rendering. Natick, Mass. : A K Peters, c2001.
- Gortler, Steven Jacob. 2012. Foundations of 3D computer graphics. Cambridge: MIT Press.

Gröller, S. Bruckner and M.E. 2007. "Enhancing Depth-Perception with Flexible Volumetric Halos." *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 6, (IEEE Trans. Visualization and Computer Graphics, vol. 13, no. 6, 2007.).

J. Kruger, J. Schneider and R. Westermann,. 2006. "ClearView: An Interactive Context Preserving Hotspot Visualization Technique." *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5 941-948.

Macaulay, David. 1973. *Cathedral: The Story of Its Construction*.

Miller, George A. 1956. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." *The Psychological Review*, 63: 81-97.

Miller, Robert. 2011. "User Interface Design and Implementation." Massachusetts Institute of Technology: MIT OpenCourseWare. <http://ocw.mit.edu>.

Moyer, Ilan Ellison. 2013. "A Gestalt Framework for Virtual Machine Control." Master of Science in Mechanical Engineering Thesis. MIT.

Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, Takeo Igarashi. 2014. "Pteromys: Interactive Design and Optimization of Free-formed Free-flight Model Airplanes." *ACM SIGGRAPH*.

Volker Blanz, Michael J Tarr, Heinrich H. Bühlhoff. 1999. "What object attributes determine canonical views?" *Perception* 28: 575-599.

