©Copyright 2013
Ankit Gupta

Interactive Playspaces for Object Assembly and Digital Storytelling

Ankit Gupta

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Brian Curless, Chair

Michael Cohen, Chair

Dieter Fox

Program Authorized to Offer Degree:
Department of Computer Science and Engineering

University of Washington

Abstract

Interactive Playspaces for Object Assembly and Digital Storytelling

Ankit Gupta

Co-Chairs of the Supervisory Committee:
Professor Brian Curless
Computer Science and Engineering

Dr. Michael Cohen
Microsoft Research

Today we observe a consistent shift towards doing our tasks virtually through machines. This mode of work ensures that the users are not tied by lack of resources required for the task, and get additional advantages like ability to make quick corrections and share the result remotely. Researchers in the field of human computer interaction have constantly pushed towards tangible user interfaces which allow the users to get a sense of doing the task physically while it happens virtually.

Designing interfaces for 3-dimensional tasks poses interesting challenges. The traditional desktop or tabletop setups do not work very well here because it is hard for the user to visualize the 3D virtual world using 2-dimensional displays and control them using un-intuitive devices like keyboard/mouse/joystick etc. Researchers and industry have explored augmented reality-style or immersive environments-based interfaces to let users interact with the virtual world. However, most of these interfaces are too specialized and hard to set up.

In this thesis, I explore an easy to set up interactive environment, called a playspace, for a variety of 3D tasks. The user performs the task while a color+depth camera observes and understands the task in real-time. It then presents context-specific feedback and automatically reflects the inferred activity in a virtual world on a screen in front of the user. The playspace also integrates other input modalities such as gestures, voice commands and standard devices like keyboard and mouse. The modular nature of the framework allows different applications to plug into the playspace environment easily.

Playspaces allow users to do a task physically while it is virtually replicated on the fly. The virtual result can then be post-processed or edited in real-time, again through physical props. The framework also opens the opportunities to assist users in real-time. I have developed and evaluated three applications in the playspace environment –

1. Block model assembly - The system automatically learns and builds a virtual replica of a Duplo® block model by observing the user build it. It also assists the user in creating a predefined model in a novel way while detecting any mistakes and assisting in making any corrections on the fly. I report on a user study that shows that the proposed guidance method is better than the traditional figure-based guidance method.

2. Digital storytelling - The system allows a user to act out a story using rigid puppets and automatically converts that into an animation. Further, it also allows the user to record multiple takes for the same story and merge them automatically after the user has roughly annotated them based on his liking. This is helpful when the user wants to try out different styles and later merge them. I report on a user study to test this utility.

3. Designing 3D environment prototypes - The system allows the user to easily manipulate virtual objects in a scene by "attaching" them to a physical object of user's choice. The user can add, move, scale, clone or delete objects from a database, thus creating simple 3D virtual environments. The user can also paint the terrain in the virtual world by using textures from his surroundings.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to thank my advisors Prof. Brian Curless and Dr. Michael Cohen for nurturing me like their own child over all these years. With a nice blend of toughness and love, they helped me grow as an independent researcher. They taught me the importance of identifying the correct questions first before marching towards finding the answers. Striving for perfection while prioritizing things is an important lesson that I will carry with me throughout my life. Thank you Brian and Michael for being patient with my mistakes and guiding me through this tremendous journey. I would also like to thank Prof. Dieter Fox and Prof. Paul Berger for serving on my supervisory committee and providing useful feedback at different stages of this dissertation.

I was fortunate to work with a number of wonderful collaborators during the course of my PhD – Pravin Bhat, Mira Dontcheva, Oliver Deussen, Larry Zitnick, Neel Joshi, Dieter Fox, Robin Held and Maneesh Agrawala. Every one of them brought their unique perspective and working style that I admire and respect. I would also like to thanks my undergraduate advisors – Prof. Subhashis Banerjee and Prof. Prem Kalra for introducing me to these wonderful fields of computer vision and graphics.

I would like to thank the UW Department of Computer Science and Engineering for providing such a conducive environment for academic growth. The top class infrastructure combined with great professors, peers and staff indeed make it one of top computer science departments in the world. In particular I would like to thank Stephen Spencer and Lindsay Michimoto for always being there for administrative or infrastructure support during my stay at the Paul Allen Center.

Hindu Swayamsewak Sangh (HSS) and HinduYUVA has been a significant part of my life at UW outside the lab. Interaction with swayamsevaks, karyakartas, vistaraks and pracharaks have always inspired me to maintain a holistic approach towards life. The HinduYUVA group gave me tremendous opportunities for self-development. I would love to continue my association with them for life.

During the course of my stay at University of Washington, I was fortunate to interact with a broad spectrum of the student community, many of who became my close friends. I feel blessed to have these friends circle who supported me through difficult times and inspired me to explore new things in life all the time. Thank you friends. I cherish every moment that we spent together in all these years.

Last but not the least, I thank my family (Mummy, Papa and Swati) for their unconditional love and support. They are, hence I am.

DEDICATION

to my parents

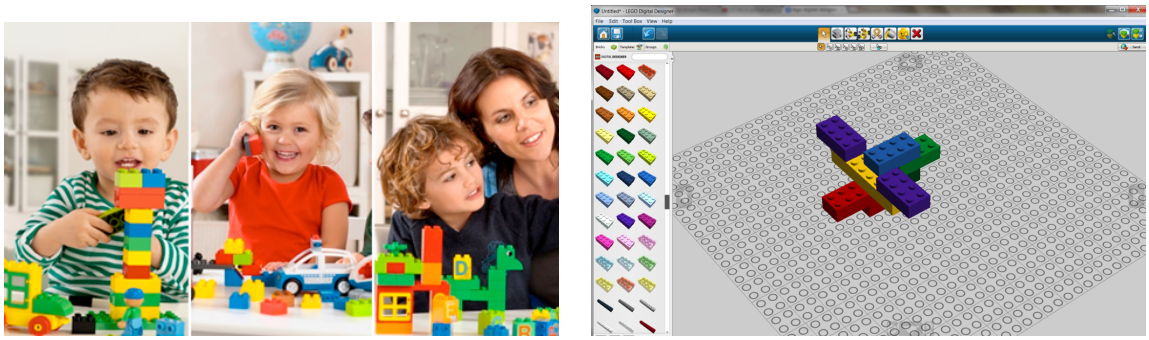
Chapter 1

INTRODUCTION

"Computers are magnificent tools for the realization of our dreams, but no machine can replace the human spark of spirit, compassion, love, and understanding." – Louis Gerstner

Computers are capable of performing a lot more mathematical operations than an average human being. These machines were traditionally used for performing complex computational tasks for scientific exploration. As their size decreased with the advances in silicon technology, the era of personal computing began to grow. Initially, personal computers were capable of doing simple tasks like viewing digital photographs, editing documents and accessing internet. Since then, immense advances in the hardware technology and human-computer interaction techniques have made these machines much more accessible and useful to us. Today, personal computing devices like laptops, phones and tablets are becoming great assistants to human beings. We are witnessing an increasing shift from doing a lot of tasks by hand (*physically*) to doing them digitally (*virtually*), e.g. preparing documents/presentations, messaging, game play, paintings, 3D modeling, etc. Key reasons behind this shift are the advantages that virtual tasks offer –

- *No limitation of having all the required physical resources, since they can be simulated. E.g.: playing the game of bowling and many others using a Wii® mote.*
- *Ease of modification/correction during the task or in a post-processing phase. E.g.: Editing digital photographs for better color settings or cropping out unwanted parts.*
- *Portability and ease of sharing the result. E.g.: Putting up a digital painting for sale on the online websites.*
- *Assistance from online knowledge bases can be seamlessly built into the system. E.g.: spell-check for documents.*



(a) Modeling with physical Lego® blocks.

(b) Virtual modeling with Lego Digital Designer® tool.

Figure 1.1: Building models with physical Lego® blocks is much more fun and intuitive than using a keyboard and mouse to click and drag blocks into place in a software-based tool like Lego Digital Designer®.

These advantages make a strong case for doing all our tasks virtually. Researchers have developed human-machine-interfaces allowing users to do the tasks virtually. However these interfaces tend to differ from the way users would do the same tasks in the physical world. Researchers have tried to minimize this gap by developing interfaces that do not involve the traditional keyboard and monitor style interaction. Instead, users work with a stylus or even their hands to manipulate 2D virtual content on a surface [56, 11, 24, 55, 83, 81, 84, 13, 75]. Thus, users can now do many 2D tasks like writing, sketching and moving widgets around in a natural way while getting all the advantages that virtual tasks offer.

Unfortunately, the current interfaces for working with 3D content are not quite as natural. Figure 1.1b shows a screenshot of the Lego Digital Designer® system for making block models. The user uses a mouse for clicking, dragging and placing blocks in a 3D virtual space. This is highly non-intuitive compared to building the model with physical blocks as shown in Figure 1.1a. Such non-intuitive interfaces are also common for tasks like 3D modeling, animation design and 3D navigation. For this reason, it is hard for lay users to work with software tools to create virtual 3D content.

The goal of this dissertation is to make it easier for the users to create 3D content. The central idea is to remove the intermediate layer of a software tool that a user would traditionally have to

use to access the virtual content. Instead, a better way is to observe the user work with physical 3D content and then use technology to transfer that into the virtual world automatically. Hence the user gets to create the content both in physical and virtual ways and enjoys the advantages of both these modes. Based on this idea, some researchers have developed immersive environments and augmented reality-based systems [41, 48, 2]. However, many of these systems are too application-specific and at times require sophisticated hardware which may not be easy for a lay user to set up.

Microsoft's Kinect® for game play serves as a good example of what a natural interface for working with virtual 3D content could look like. Instead of using buttons to control the 3D digital avatars, the camera tracks the human body pose [75] and maps the motion to the avatar. Thus the user can literally walk around and jump in his living room while making his avatar do the same for game play. The product has received rave reviews from people all around the world who claim that this interface makes them *play* the game rather than *control* the game. Currently this system is specific to tracking human bodies and cannot be generalized to work with different types of 3D content.

In this dissertation, I design and explore a more general framework in a slightly different setting. It allows the user to do a task physically on a planar work surface, while the task is replicated virtually using a sensing mechanism. The virtual counterpart can be appropriately modified at any stage and can be easily shared across different users or setups. The framework also allows the user to use physical proxies for high fidelity virtual counterparts. In addition to this, it allows for back and forth feedback between the virtual and physical world using different modalities. More importantly, the framework is modular in nature, enabling different applications to be plugged into the same setup easily. Thus, the playspace framework aims to achieve all the advantages of doing a task virtually while still allowing the user to do it physically.

In the following section, I describe the concept of a playspace in more detail and a few applications that I use it for. Finally I conclude the chapter by outlining the organization of this dissertation.

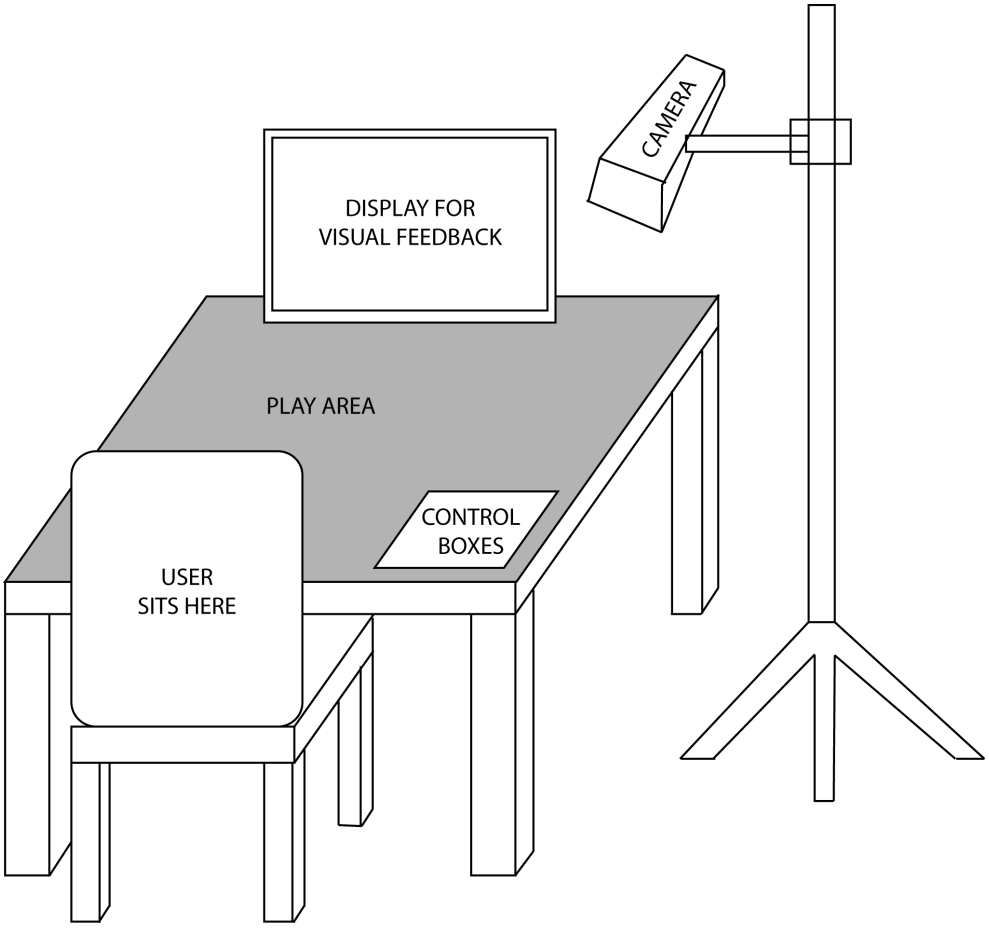
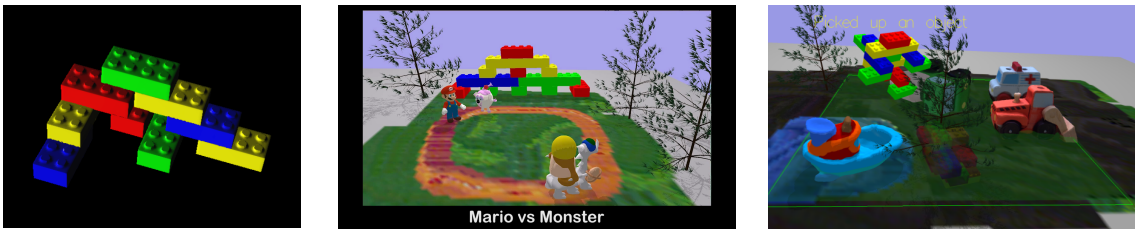


Figure 1.2: Playspace organization: User works on a planar work surface which is divided into two regions. The *Play Area* is mapped to a part in the virtual world which is rendered on the screen in front of the user. Any physical object manipulations in the *Play Area* are tracked by the camera and reflected in the virtual world on the screen. *Control Boxes* are volumetric regions which can be used by applications as check boxes, buttons or sliders using physical props as handles. The playspace also integrates input through voice and standard devices like keyboard and mouse.

1.1 Playspace

A *playspace* is an interactive system that aims to combine the advantages of doing a task physically and virtually. Figure 1.2 shows the overall organization of a playspace. In this setup, the user works on a planar work surface. The surface is divided into two parts – *Play Area* and *Control Boxes*. Any physical objects in the *Play Area* are tracked in real-time using the Kinect® color+depth camera. The *Play Area* is exactly mapped to a part of the virtual world which is rendered on the display



Block Model Assembly | Digital Storytelling | 3D Scene Design

Figure 1.3: The dissertation presents novel interactive systems for virtual 3D content design applications – Block model assembly (Chapter 3), Digital storytelling (Chapter 4) and 3D Scene design (Chapter 5).

screen in front of the user. The tracked motion of physical objects is reflected in the virtual world in real-time. The *Control Boxes* can be used for gesture-based inputs. Further the playspace integrates other input modalities like voice-based commands or keyboard and mouse inputs. I would like to emphasize that the playspace is a general framework in which different applications can be plugged in. I describe the process of setting up a playspace in detail, in addition to the algorithms for analyzing various input modalities, in Chapter 2.

In this dissertation, I also explore the use of playspaces for three different applications, broadly related to assisted creation of 3D content - block models, animations and 3D scenes. The existing software-based tools for these applications have a steep learning curve and hence their usage is limited to expert users. I strongly believe that such tools for expressing one's creativity should be accessible to lay users even if they trade off content fidelity with accessibility. I demonstrate that playspaces can be utilized for these applications to enable novel interfaces for expressing and sharing one's creativity.

I now briefly describe the three applications. Each of them has novel algorithmic contributions related to real-time analysis of the camera feed and providing context-specific assistance to the user.

(a) **Assembly of Block Models.**

Building block models with Lego® or Duplo® blocks is a popular hobby across adults and children. The block sets usually come with a set of instructions to put together a preconfigured

model. These instructions can be hard to understand at times for users and any undetected mistake can require backtracking and re-doing a lot of steps. Instead, it will be better to have technology detect any mistakes and automatically and assist the users to make corrections.

I propose a system *DuploTrack* [27] where a user works in a playspace with physical Duplo® blocks to build a pre-defined model. The system uses a novel 3D-tracking based guidance method to present instructions to the user. It also tracks the assembly process in real-time, points out any mistakes and helps correct them. The capability to track the assembly process also enables the system to learn how a new block model is assembled by a user. This learned representation can be used to share the model with other users via automatically generated representations like virtual 3D mesh models, static instructions, instruction videos or by bootstrapping it back into the system for guiding a new user.

DuploTrack is a novel system which guides a user to build pre-configured block models as well as automatically learn the assembly of a new model by watching a user build it. In this work, I also report on a user study which compares the traditional figure-based guidance methods to DuploTrack's guidance method. DuploTrack forms a basis for developing more sophisticated algorithms in the future for complicated tasks like furniture assembly, fixing bike parts, changing a printer cartridge etc.

(b) **Digital Storytelling.**

Animations are a great way to tell a story. However, the current tools for creating animations like Maya® have a steep learning curve which may prohibit a lay user from using them. I propose a novel puppeteering-based interface for creating digital 3D animations using playspaces. The first part of this work, *3D-Puppetry* [30], was done in collaboration with Robin Held and Maneesh Agrawala from UC Berkeley. 3D-Puppetry allows users to act out stories using physical objects in a playspace. These stories are tracked in real-time and are converted to digital animations on the fly.

From our user studies with 3D-Puppetry, we learned that lay users often do not have the same artistic knowledge about depicting animated motion as trained animators. Hence they often record different motion depictions of the same story and then choose the one which they like

the best. However, it is common that the user likes or dislikes different parts of the different takes, and it is hard to achieve the perfect story in repeated attempts.

I present a system *MotionMontage* which allows the users to record multiple takes of a story and roughly annotate the parts based on their liking. I then propose a novel algorithm to combine these annotated takes into a *montage* which preserves the users' annotations and maintains the temporal continuity. This allows the user to achieve a *perfect* animation with a few *imperfect* takes. I also report on a user study which shows that users find the system easy to use and that the montage is perceived significantly better than the individual takes.

(c) **Design of Virtual 3D Scenes.**

Virtual 3D scenes are commonly designed by animators for animated movies, games, advertisements etc. As with digital animations, the software tools have a steep learning curve and hence are not very accessible to lay users wanting to express their creativity. Games like Minecraft® allow players to build virtual scenes by placing and moving simple block primitives. These simple yet powerful games motivate the need for research towards more intuitive interfaces for lay users to design 3D scenes.

The key limitation of the existing interfaces is that the motion of the input device does not map directly to the 3D positioning and motion in the virtual world. Hence I propose a system where the working 3D volume of the playspace is mapped to a volume in the virtual scene and the user can use any 3D physical object as a *controller*. The motion of the controller is directly mapped in the virtual scene and hence the user has a better 3D perception about the scene. The controller can be used to move, resize or delete existing objects in the scene or add new objects from a database. Again, different input/output modalities of the playspace have the potential to make this a simple, intuitive, and powerful interface to quickly design virtual 3D scenes.

This work is still at a primitive stage. Ideally, the user would be able to use simple proxies in the *Play Area* of the playspace to arrange a 3D scene. These 3D scenes can then be used for digital storytelling or as prototypes for professional designers to work on. This system can also be extended to collaborative scene designing systems. There is a lot of related research in the

field of 3D design and modeling which can be integrated to improve the user experience and the quality of the results.

1.2 Organization of the Dissertation

In this dissertation, I propose and explore a novel *playspace* framework that allows the users to perform tasks both physically and virtually in a seamless way. The framework provides a number of input modalities to the users and allows for plugging in different applications easily. I describe the hardware setup and the underlying software framework of a playspace in Chapter 2. Then I demonstrate the efficacy of playspaces with three applications – *Assembly of Block Models* (Chapter 3), *Digital Storytelling* (Chapter 4) and *Design of Virtual 3D Scenes* (Chapter 5). Each of these chapters include a discussion about the prior research in these domains, detailed description of the corresponding algorithms and user interaction design, performance analysis through user studies, and current limitations. I conclude this dissertation in Chapter 6 by summarizing the contributions and discussing future work directions.

Chapter 2

SETUP AND ALGORITHMS FOR PLAYSPACES

"It's not enough that we build products that function, that are understandable and usable, we also need to build products that bring joy and excitement, pleasure and fun, and yes, beauty to people's lives." - Donald Norman

2.1 Overview

A playspace allows the user to interact with a virtual world that is rendered on a display screen. The interaction takes place through multiple modalities. Figure 1.2 shows the organization of a playspace. The planar work surface is demarcated into two regions – the *Play Area* and *Control Boxes* – and a Kinect® color+depth camera looks down at the table. A part of the virtual world is directly mapped to the *Play Area* and any physical objects in this region are tracked in real-time. The user can manipulate the physical objects to reflect the changes exactly in the virtual world on the screen. The playspace also allows for more input modes – gestures in *Control Boxes*, voice commands and devices like keyboard and mouse. Any application running in a playspace analyzes the input modalities and provides a context-specific real-time visual feedback on the display screen superimposed on the virtual world.

Figure 2.1 shows the software framework of the playspace. The streams from the input modalities are given as input to the playspace algorithms –

- RGBD Processing module for camera feed. The PCL library [67] is used to access the Kinect® sensor's camera color+depth feed which is analyzed using computer vision algorithms. I describe these algorithms in detail in Section 2.3.

- Voice Recognition module for microphone feed. Voice recognition is done using the Julius library [46] for Linux platform. The application defines a grammar which is given as input

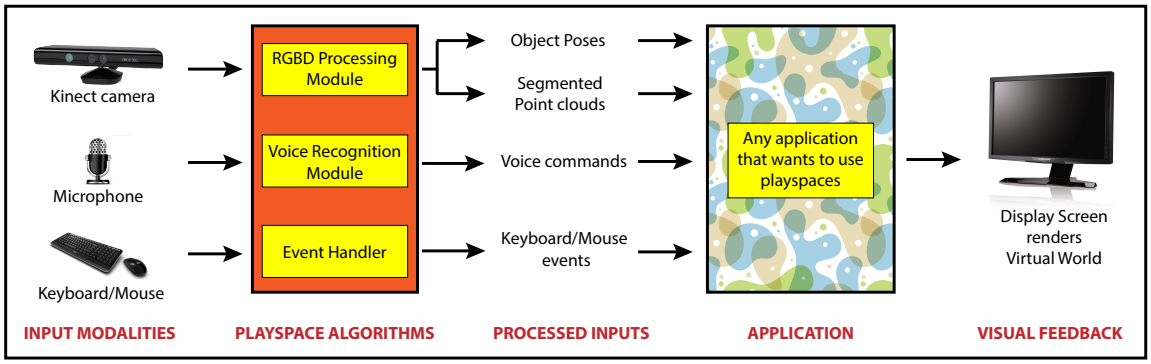


Figure 2.1: Software framework of a playspace. The streams from the input modalities are given as input to the playspace algorithms – RGBD Processing module (for camera feed), Voice Recognition module (for microphone feed) and Event handlers for keyboard and mouse. The outputs of these algorithms are given as controls to the application running on the playspace. The application renders the virtual world and provide context-specific visual feedback on the display screen.

to the library. The library is fairly robust for American accents and also provides confidence values for the detected words which can be used by the application for robustness.

- Event handlers for keyboard and mouse. These handlers are implemented using in-built call-back functions in OpenGL [70].

The outputs of playspace algorithms are given as inputs to the application running on the playspace. The application renders the virtual world and provides context-specific visual feedback on the display screen. The OpenGL library [70] is used for 3D rendering on the display screen. All the components of this framework are implemented to run in parallel on the Ubuntu-based ROS platform [62].

In the next section, I explain how a playspace is set up initially. This involves setting up the hardware and some preprocessing steps for initializing the playspace algorithms. Next, I describe the computer vision techniques for processing the camera feed in detail. Finally, I conclude the chapter by summarizing the concept of playspaces.

2.2 Setup of the Playspace

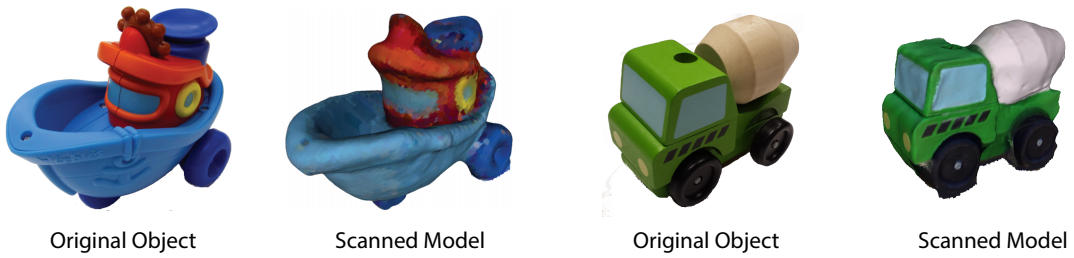
The configuration of various interface components of a playspace is flexible. This is an important feature of playspaces which makes them easy to use under various conditions. The user needs to first set up the hardware – work surface, the color+depth camera looking down at the work surface and the display screen. He then scans the physical objects that need to be tracked in the Play Area. Next, he defines the *Play Area* and the *Control Boxes* on the work surface and the system learns to segment out the camera feed based on that. Finally the system learns a color model for the user's skin via a machine learning algorithm. I now discuss each of these components in detail.

2.2.1 Hardware setup

Figure 1.2 shows a possible hardware setup for the playspace. The user first needs a planar work surface which can be a table-top or floor etc. The Kinect® camera needs to be positioned in such a way that it looks down on the work surface. The camera positioning has a couple of caveats. First, it is better if the camera's view direction is at an angle of about 20 to 40 degrees with the normal of the work surface. This allows the camera to have a better geometrical view of the work surface. Second, the height of the camera should not be too low since the depth sensors have a limited working volume and cannot sense very close data. The Kinect® sensor does not provide reliable data for depths closer than $0.5m$. The camera should be sufficiently high to closely cover the work surface in its field of view. This allows it to use most of its resolution for relevant pixels. The user can sit on one end of the working surface and place the display screen right across on the other side to get a good real-time view of the virtual world while the task is done. Other devices like keyboard, mouse, external microphone can be placed where it is convenient for the user to use them. All the devices are connected to a computer on which the algorithms run in real-time.

2.2.2 Scanning the Physical Objects to be Tracked

The playspace tracks the 3D pose of rigid objects in real-time in the *Play Area*. The pre-requisite for this is having the virtual 3D scans of the objects, also denoted as virtual models. There are many off-the-shelf software tools which allow us to scan objects using depth sensors (e.g., KinectFusion [35] or ReconstructMe® [28]) or color sensors (e.g., AutoDesk® 123D Catch® [9]). These algorithms



(a) Boat. (b) Truck.

Figure 2.2: Software tools like ReconstructMe [28] can create rough 3D scans of objects by moving a Kinect® around them.

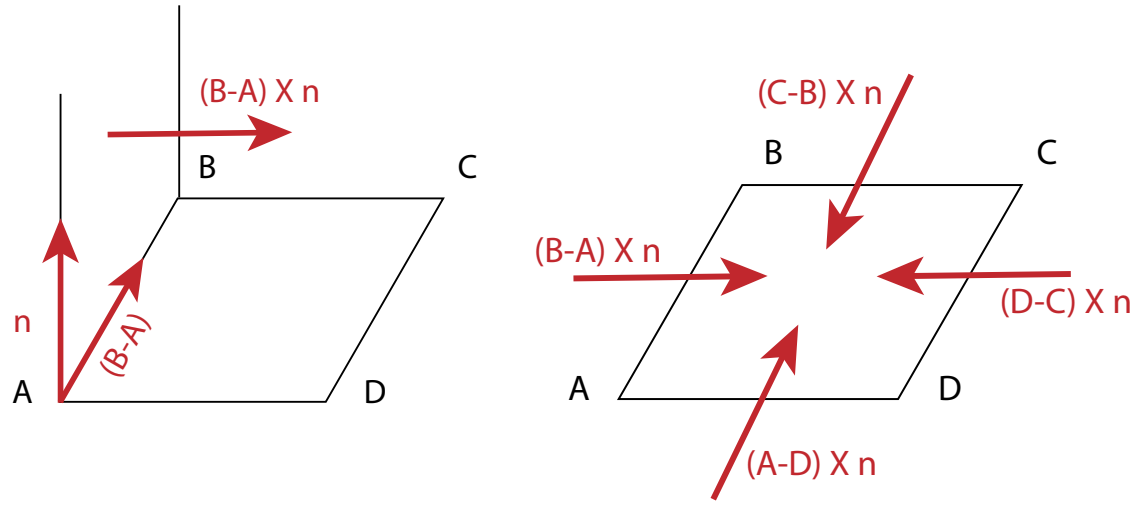
rely on fusing multiple depth and/or color images and work fairly well. Figure 2.2 shows a couple of examples of scans obtained using the ReconstructMe [28] tool. The scan quality is still not perfect and will improve as the sensors and fusion algorithms improve. The users can also use expensive laser scanners or existing CAD models to obtain high fidelity virtual models.

The current implementation needs the virtual models in the PLY format for tracking. I use MeshLab [80] to convert the different file formats to PLY format. I also compute the per-vertex normals for the virtual models using MeshLab. These are used in the tracking algorithm which is described later.

2.2.3 Defining the Play Area and Control Boxes

The work surface of a playspace has a *Play Area* in which the physical objects are tracked, and a set of *Control Boxes* which can imitate toggle buttons or sliders or can be used for gesture recognition. The number of *Control Boxes* required can vary with the application that uses the playspace. For the purpose of this discussion, I consider the *Play Area* and *Control Boxes* together as a set of volumetric boxes bounded at the bottom by the planar work surface and height parallel to the plane normal.

On start up, the system fits a 3D plane to the work surface using a RANSAC-based technique [23]. The assumption here is that the work surface forms a majority plane in the camera's field of view which was also suggested during the hardware setup. The output of the Kinect® camera is a 3D point cloud. RANSAC considers sets of 10 randomly chosen points from the point cloud



(a) Computing normal for one wall.

(b) Normals for all the walls of the *Play Area*.

Figure 2.3: The volume for *Play Area* is computed by first asking the user to click four corner points in clockwise order. These four points define the bottom surface of the volume for *Play Area*. The system assumes that the *Play Area* extends perpendicular to the work surface, i.e. bounded by four walls in the direction of work surface's normal n. The system computes a normal for each wall of the *Play Area* such that the normal points inside the volume. These normals can be used to check if a 3D point inside the volume or not.

and fits a plane using a standard least squares formulation [63]. The algorithm repeats this for different random sets of points and chooses the plane which gets the maximum support from the 3D point cloud, i.e. the number of 3D points which vote for lying on that plane. Mathematically, this plane is denoted by a 3D point (P) and the normal vector (n).

The user can define each box by a simple process. The user is shown a screenshot from the Kinect®'s viewpoint. He clicks four corner points in clockwise order for the bottom surface of each box that needs to be defined. The corresponding four points in 3D are used to define the four planar surfaces (walls) perpendicular to the work surface, thus bounding the box on the sides. Let the four points for the *Play Area* be ($\{A, B, C, D\}$) as shown in Figure 2.3. The first wall connecting A and B is a plane containing these points whose normal vector is $(B - A) \times n$. Similarly we can define the planes for all the four walls for the box. We can use this to classify whether a given 3D point is

inside a box or not. A point is inside a box if it lies in the positive half-spaces of the normals for all the four walls and the bottom surface. The direction relative to the normal of a plane can be checked by taking the dot product of the plane's normal vector with the vector from a point on the plane to the 3D point. A positive value indicates along the direction and opposite otherwise.

Hence, the user defines all the boxes in the work surface (*Play Area* and *Control Boxes*) and the system computes their volumetric representations. This will be used in the segmentation step described later to discard any pixels that do not lie in any of the boxes.

2.2.4 Learning Color Model for User's Skin

The system needs to segment out the user's hands in the *Play Area* for tracking the objects. This segmentation is done using a skin-color-classifier learnt at this step. After defining the *Play Area*, the user runs the module for learning the skin color and waves his hands in the *Play Area*. The system segments out the hand pixels from a window of 10 frames using the background subtraction technique (discussed in the next section) and builds an aggregate histogram using those pixel colors in HSV color space. The color channels lie in the range of 0.0 to 1.0 and each channel's histogram bins are of the size 0.2. The color bins having more than 10 pixels are chosen to be skin color bins. The learned color model is used for skin color classification. If a pixel's color lies in a skin color bin after conversion to HSV color space and quantization, it is classified as user's skin. Otherwise the pixel is considered part of the object to be tracked.

There has been a lot of research for building general skin color models [40]. However, there is no standard color model that works well for all users. Since we need a reasonably high classification rate and our system is already interactive, I use the per-user approach to learn a skin color model separately for every user. The color models for the users can be saved over time and need not be learnt again if the lighting conditions do not change considerably.

2.3 RGBD Processing Module for Kinect® 's Color+Depth Feed

Figure 2.4 shows the processing pipeline for the feed from the Kinect®. The Kinect® camera provides a continuous stream of images with an RGB color and depth at each pixel. The *Segmenter* module takes each color+depth image from the camera and prunes it to the pixels corresponding to

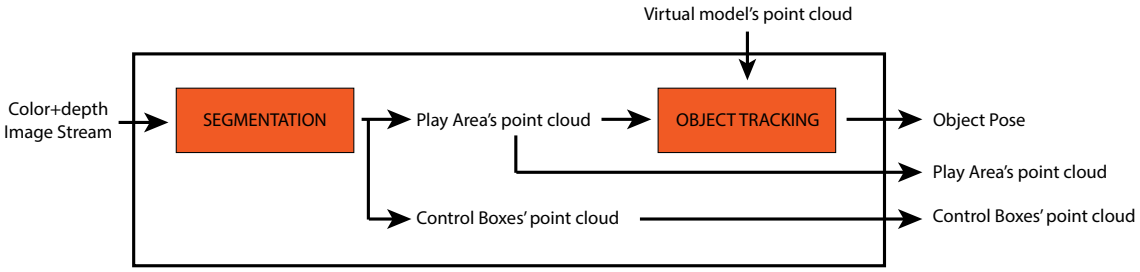


Figure 2.4: The RGBD Processing module receives color+depth images from the Kinect® camera. It first segments each image to extract the foreground pointclouds in the *Play Area* and the *Control Boxes*. It tracks the object in the *Play Area* using the its virtual replica's pointcloud. The pose of the object in the *Play Area* and the two foreground clouds are returned as outputs.

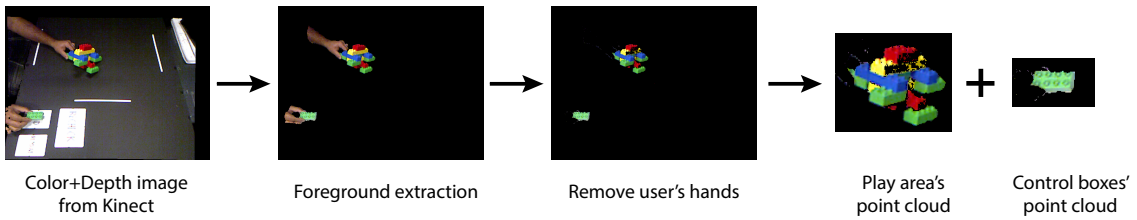


Figure 2.5: The segmentation algorithm works in two stages. In the first stage, it uses background subtraction followed by a test of inside *Play Area* or *Control Boxes* to extract the foreground. In the second stage, it uses the skin color model learnt at start to remove the pixels corresponding to skin. It gives out the 3D point clouds corresponding to the remaining pixels in the *Play Area* and the *Control Boxes*.

the physical objects in the *Play Area*. It passes the corresponding 3D point cloud, denoted by P_c, to the *Tracker* module. It also extracts the pixels in the *Control Boxes* and converts that to a point cloud which forms one of the outputs. The *Tracker* module tracks the physical objects and provides their poses as an output. I now describe the segmentation and object tracking algorithms in more detail and discuss their limitations.

2.3.1 Segmentation

The segmentation algorithm receives a color+depth image and returns two point clouds corresponding to (a) the physical objects in the *Play Area* and (b) the foreground pixels in the *Control Boxes*.

The algorithm works in three stages as shown in Figure 2.5 –

1. Extracting the foreground in *Play Area* and *Control Boxes*.

   First, the system removes background pixels via background subtraction. The system starts with an empty work area and stores the background depth for each pixel. For subsequent images, for every pixel in the incoming depth image, if the observed depth is less than 95% of the background depth, that pixel is marked as foreground. The 95% value is empirically determined to counter the depth inaccuracies from the Kinect® camera. This simple test removes a lot of background clutter and leaves the pixels that correspond to any moving background or relevant parts on the work surface.

   Next, the system removes all the pixels whose corresponding 3D points are not inside the *Play Area* or any of the *Control Boxes*. The 3D point for a pixel can be obtained by backprojecting the pixel using its observed depth value and the camera's internal parameters which are known beforehand. Checking whether this point is in a 3D box can be done as described earlier in Section 2.2.3. After doing this for all the pixels, the remaining pixels correspond to user's hands and physical objects in the *Play Area* and the *Control Boxes*. The 3D points corresponding to the pixels in the *Control Boxes* are directly given as the output without further processing. The remaining pixels from the *Play Area* undergo one more filtering stage to remove the pixels corresponding to the user's hands.

2. Removing Skin-color from the *Play Area*.

   The stage uses the learned color model for the user's skin, described in Section 2.2.4, to remove any pixels in the *Play Area* whose colors lie in the skin color bins after being converted into the HSV color space. The remaining pixels in the *Play Area* correspond to the physical objects and the corresponding point cloud is passed on to the Tracker module for pose estimation.

Performance and Limitations

The depth-based background subtraction is observed to work very well for removing the static background clutter. Ideally, the order of the first two stages can be switched. However, doing background

subtraction first helps because it involves lesser floating point operations per pixel compared to checking a 3D point for lying inside volumetric boxes.

The skin-color segmentation algorithm has problems when the color of the physical objects is close to the user's skin. This leads to some parts of the physical objects getting segmented out at times. One straightforward, if not elegant, solution to this is to make the user wear special colored stockings on the hands. However, there are a couple of solutions that might work better. Heat sensors have been used by researchers [21] to detect and track human body parts and can be useful in this case too, albeit at the cost of an additional sensor. Another solution might be to use an algorithm which fits a user's hand and forearm in the scene and then tries to segment it out. I leave these ideas for future research.

2.3.2 Object Tracking

A key component of the system is the ability to track the physical objects in the *Play Area*. In this section, I will only discuss tracking of one rigid object using the segmented point cloud from the Kinect®. Tracking multiple objects was done as a joint work with Robin Held for the 3D-Puppetry system [30] and does not form the part of this dissertation.

As described in Section 2.2.2, we need to have the virtual 3D model of the physical model to track it. The virtual model's point cloud is referred to as P_v and the segmented point cloud of the physical object as P_c for the purpose of this discussion. Tracking the object essentially means that P_v needs to be aligned with P_c. The system uses the ICP (Iterative Closest Point) algorithm proposed by Besl and McKay [12] to align the point clouds by solving for a 6D transformation: 3D rotation and 3D translation.

Given an initial transformation T_0 between the point clouds, the ICP algorithm iteratively solves for the final transformation T_f. The algorithm first applies T_0 to P_v to create a transformed virtual point cloud. For each point in P_c, it finds the closest point in the transformed P_v, accelerated with a K-d tree. Correspondences with distances greater than an outlier threshold are rejected. The transformation is then updated by minimizing the total squared distances between the remaining correspondences. This optimization is solved in closed form. The same steps are repeated with the updated transformation until convergence. At each iteration, I reduce the outlier-rejection threshold.

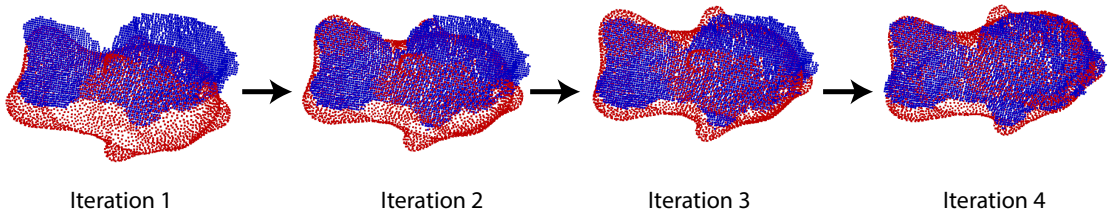


Figure 2.6: The ICP algorithm [12] for object tracking computes a transformation that aligns the red point cloud (virtual model) to blue point cloud (physical object). In each iteration, the algorithm finds nearest neighbor correspondences between clouds and computes a transformation to align the correspondences. This process is repeated till convergence and the transformations are sequentially combined to compute the final transformation that aligns the two clouds.

Figure 2.6 shows a few iterations of the ICP algorithm to align the red point cloud P_v to the blue point cloud P_c.

Since the ICP algorithm performs a greedy, local optimization, success depends on good initialization, T_0. Typically, T_0 is set to the T_f from the previous tracked frame. However, when the object first appears before the camera, there is no previous transformation available. Further, the system can lose track of the object, e.g., when the motion is too fast. We detect loss of tracking when the distance between the nearest point correspondences becomes too large. Thus, when tracking is lost or a model is newly introduced, we require a method for *pose initialization*.

The initial pose can be computed by matching visual features, for instance SIFT [51], of the observed physical object to a pre-compiled feature database of different views of the object. However, the visual features do not work with objects which lack sufficient texture. Hence in this dissertation, I use a 3D geometry-based method for initializing the pose. In the 3D-Puppetry system [30], we combine the feature-based and geometry-based tracking algorithms to robustly estimate the pose. I refer the reader to that paper [30] for more detail about this hybrid method.

Pose Initialization

To initialize the pose, we need to estimate a suitable 3D translation and 3D rotation. For 3D translation, I simply use the difference between the 3D centroids of P_v and P_c. To estimate an initial rotation, I use a brute force approach to generate all possible orientation candidates to align P_c and

P_v and evaluate them in parallel to find the best alignment. The parallel computation is done using the pthreads library [58]. To avoid finely sampling all possible rotation values, I make use of the fact that ICP algorithm can align two point clouds if they differ slightly in alignment. Hence I sample each rotation dimension at the resolution of 10 degrees which leads to a total of few thousand candidates.

Then, for each orientation candidate j, the virtual model point cloud P_v is transformed using the corresponding candidate rotation as well as the previously estimated translation. The visible parts of the transformed P_v are selected using a depth buffer rendering from the camera's viewpoint. Then the ICP algorithm is used to refine the orientation to arrive at a candidate transformation T_j. Applying T_j to P_v results in a candidate point set P_j to match against samples P_c from the Kinect®. The best transformation candidate T_j having the minimum match error between P_c and P_j is chosen as the initial pose.

For the match error, one simple option is to use the proportion of points in P_c which do not find a close match in P_v. However, it is hard to decide a 3D distance threshold for classifying a match as close or not. Hence I use the difference in the projections of the two aligned point clouds as their match error. This metric is observed to work better in general. The system renders the colored point clouds P_c and transformed P_j from the camera's viewpoint to create two color images. First, the images are partitioned into abutting 50×50 windows. For each window, a color histogram is built by clustering the pixels into the bins of corresponding colors. The pixels in P_c's projection may have saturated pixels which cannot be classified as any color. I put them in a separate bin called saturated-color bin. I set the match error between the images as the sum of the match costs between the histograms of corresponding windows.

I use the Earth Mover's Distance (EMD) [65] metric to match any i^{th} histograms in the two images. This metric measures the cost of recreating the latter histogram by moving around the data in former. There is a cost to move data between the bins. I set the cost to move pixels from any color to a different color as 1. I set the cost to move pixels from the saturated-color bin to any color bin as 0.

The Special case of Block Models

Many of the applications described in this dissertation work with block objects, i.e., objects composed of mutually orthogonal planar surfaces. In such cases, the rotation component of the initial pose can be chosen from a much smaller candidate space. We can estimate the dominant face normal directions, resulting in a small set of possible rotations to align P_c and P_v. In fact, two normal directions suffice, as the third direction can be inferred from the cross product of the first two.

First, normals at every point in P_c are estimated based on each point's position and the positions of its nearest neighbors. The normals are clustered into bins in direction space, i.e., bins over the unit sphere. A Hough-transform finds a pair of mutually-orthogonal bins in the direction space which have the most normals supporting them. Given this pair of directions, there are 24 possible orientations of the X, Y, and Z axes such that any two line up with this pair. Hence the 3D rotation search space can be pruned to testing 24 candidate orientations, $j = 1...24$. For tracking models made of Duplo® blocks in Chapter 3, I assume that the hollow side of the blocks does not face up since there is not enough 3D geometry to track the model.

Performance Analysis and Limitations

The performance of the object tracking algorithm depends directly on the computational complexity of the ICP algorithm. The ICP algorithm consists of two major parts that run in each iteration – (a) Nearest neighbor correspondences between the point clouds and (b) Fitting a pose transformation. The bottleneck for the algorithm is the first step of finding correspondences. Let us assume the size of the point clouds to be n, where n is the number of points. One of the clouds is stored as a K-d tree so the nearest neighbor search for one point takes $O(log\ n)$ time. Hence the total time for computing correspondences is $O(n\ log\ n)$. I employ two techniques to improve this performance. First, I run the nearest neighbor searches in parallel using pthreads library. The number of threads running in parallel is limited to the number of hardware threads available to the processor. In the future, we can also implement this on the GPUs to massively parallelize these searches. Secondly, I put an upper limit of 1000 to the size of the point clouds. If the point clouds grow bigger than that, I spatially downsample them uniformly by an appropriate factor to bring the cloud sizes under 1000.

The tracking algorithm still has some limitations. Parts of the object being tracked can get

occluded by other parts of the object itself or the user's hands. The tracking algorithm is not able to align a complete point cloud of the virtual model to that of the physical object in case of significant amount of occlusion. Occlusions can be handled by augmenting the tracking algorithm with visual feature-based tracking as shown by Held et al. [30] in the 3D-Puppetry system. Tracking small physical objects also causes errors. This is because the current resolution of the depth sensor is not high enough to provide rich geometrical data about the physical object, thus leading to errors in the alignment process. Newer sensors are seeing a significant improvement in the resolution and field of view now, and I believe using such sensors will significantly extend the tracking range of the tracking algorithm.

2.4 Summary

In this chapter, I have explained the hardware setup of a playspace and the algorithms for analyzing the input feed from various modalities. The algorithms run in parallel and in real-time. I also discussed the limitations of the algorithms and proposed how they could be overcome. The software framework of playspaces is modular in nature and allows plugging in different applications easily. In the subsequent chapters, I will discuss three applications related to creation of 3D virtual content that benefit from this framework.

Chapter 3

BLOCK MODEL ASSEMBLY IN PLAYSPACES

"You'll see more and more perfection of that - computer as servant. But the next thing is going to be computer as a guide or agent." – Steve Jobs

3.1 Introduction

Block model assembly toys have retained their popularity over time. These are particularly liked by children who start assembling models at an early age, developing spatial skills useful throughout life. Lego® and their larger cousin Duplo® blocks are well known snap-together blocks for assembling models. These blocks are designed to be easily assembled into interesting models and de-assembled for reuse.

Model assembly often follows printed step-by-step instructions as shown in Figure 3.3a. Such step-by-step instructions can be hard to follow and are not very robust to mistakes during the assembly process. Also, rebuilding a model that we created at some earlier point in time would require re-finding the instructions. Or if we want to share our original physical models with friends who want to build a replica, there are no easily generated instructions. We could save/share the completed model in its physical state but then we might run out of blocks while building more models, and even so, the completed model serves as a poor instructional device. The Lego Designer Tool® [77] allows users to create virtual models using a keyboard and a mouse and can then generate instructions for them. But making a model virtually can be far less intuitive than actually making the physical model.

Ideally, we can save the construction of the model in some digital format from which it is easy for the same user or others to rebuild it later in future. One could take photographs or video of the model during construction. Casually captured videos of the construction process may be useful in guiding the rebuilding process, but can be confusing or tedious to follow especially if there is any backtracking in the construction process. A user could also create a well-annotated set of instructions in

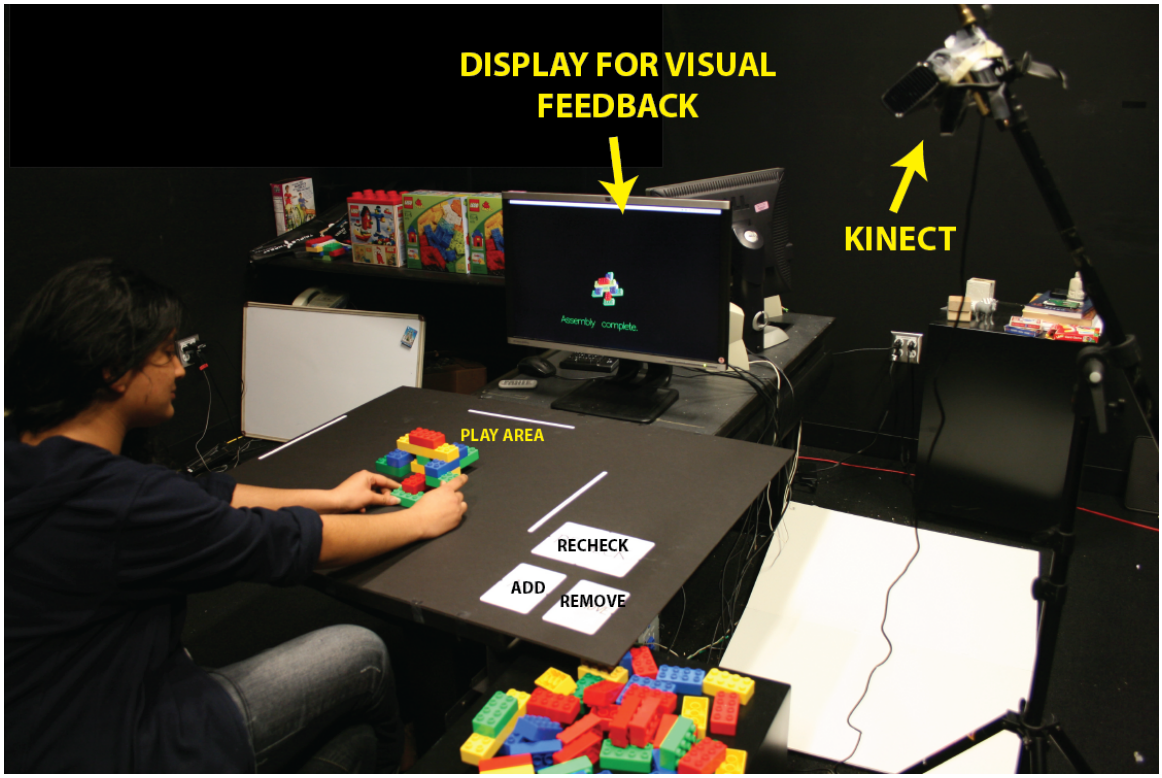


Figure 3.1: System setup. The user builds the model in the *Play Area* and uses the *Control Boxes (Add/Remove/Recheck)* to interact with the system. The Kinect® looks down from the right and passes the captured video/depth stream to our system to track the model and infer the assembly process in realtime. Visual feedback is shown on the display in front of the user.

form of photos or figures like standard Lego® blocks instructions; however, this requires great skill and significant effort. We would like to enable users to build models with minimal interference from the recording mechanism and to be able to store and transmit instructions to rebuild that model later with ease.

Motivated by these issues, I implement a system, *DuploTrack* which allows the user to assemble a block model in a playspace environment and infers the assembly process in realtime. Figure 3.1 shows a user using the system. It dynamically tracks the physical model in the *Play Area* and displays a virtual replica on the screen in front of the user in the same pose as the in-hand physical model. In *Authoring* mode, it records the step-by-step addition of blocks. Removal of blocks is also handled, effectively deleting the previous addition of that block. In *Guidance* mode, step-by-step

instructions are superimposed on the digital replica, again following the dynamically changing pose of the physical model being constructed. The system detects mistakes made and gives appropriate feedback to the user, thus avoiding the user's frustration of undoing and redoing multiple steps for making a correction in the assembly.

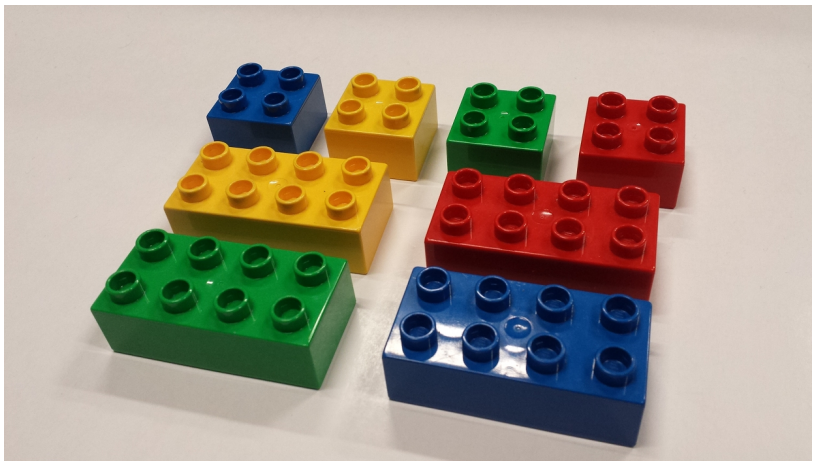
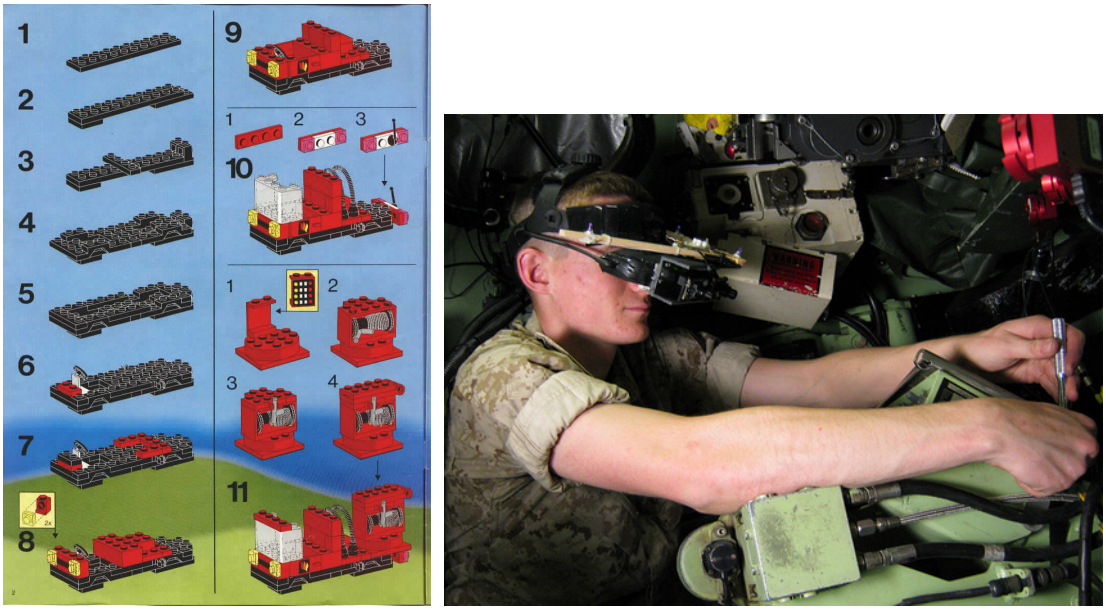


Figure 3.2: Our system uses 2×4 and 2×2 Duplo® blocks of colors – red, green, blue and yellow. These form the majority of the Duplo® Basic Bricks Set 6176.

Although blocks come in varying shapes and sizes, to keep the inference problem computationally tractable, the system works only with Duplo® blocks that each have a 2×4 or 2×2 arrangement of studs, and have colors – red, green, blue and yellow, as shown in Figure 3.2. These blocks form the majority of the commercial Duplo Basic Bricks Set 6176. I later discuss how our solution can be extended to include other different types of blocks.

In this work, I also report on a user study to answer some specific questions related to the guidance system, comparing it to traditional guidance via static images. I measure time taken to complete assembly tasks and count the number of mistakes made by users.

I explore two broad directions in this chapter – a novel way of guiding assembly process and a realtime system to track and infer the assembly process. In the next two sections, I review the prior work in these directions. I then present the design, implementation and potential applications of this system. After that, I describe the user study, analyze its results and conclude the chapter by discussing potential future work.



(a) Traditional figure-based guidance.

(b) High end augmented reality guidance [31].

Figure 3.3: Methods of guiding people for assembly tasks exist at extreme spectrums. Traditional figure-based methods provide no feedback and may not be understood equally by everyone. In comparison, the sophisticated augmented-reality based methods require a considerable hardware setup but provide continuous assistance from user's viewpoint.

3.2 Related Work

3.2.1 Guiding the Assembly

Agrawala et al. [5] provide an excellent summary of the issues in designing the presentation of many assembly tasks. Heiser et al. [29] and Agrawala et al. [6] have studied design principles for producing visually comprehensible and accessible instructions for assemblies, and develop algorithms for producing such instructions. A key observation is that creating effective static instructions for three dimensional tasks is difficult and should follow established design principles.

I now discuss some common modes of presenting assembly instructions to the users and discuss what depth perception cues they provide.

Figure-based guidance. Figures (as drawings or photographs) are the most common way of providing instructions in assembly tasks. The user is shown a figure or multiple figures depicting

the current state of the model and showing where the new block goes as in Figure 3.3a. This type of instruction is used both for toy models such as Lego® as well as in many other assembly tasks such as IKEA® furniture assembly. Each figure provides monocular depth cues such as occlusion, perspective, relative size, and shading. While all these cues can aid structural perception, the spatial perception literature [19] has shown that motion cues also play a central role in understanding shape. Motion parallax [22] and the kinetic depth effect [82] can greatly enhance the structural perception of the model. In addition, any structure that the user understands from static figures needs to be mentally aligned to the physical model in his hand. Once done, the user can add the new block. I call this perceptual alignment from virtual model to physical model *perception transfer*. Perception transfer can induce errors in structural understanding which I hope to minimize in our guidance system.

Video-based guidance. An alternative to figure-based instructions is video-based guidance, presenting the user with recorded videos of the assembly steps. The user can then pause, play or repeat each video clip to understand the instruction and then perform it. Videos can provide the same depth cues as figures, but also provide motion cues that lead to better structural perception. Video instructions can also show the motion of the parts as they are being placed which can be particularly useful if the task requires complex moves. However, the user may find it hard to control the system, needing to pause the video sometimes to understand or replay the clip multiple times. Further, the problem of perception transfer discussed for the figures still remains. Pongnumkul et al. [60] have developed a system for generating Pause-and-Play video tutorials and discuss these common problems.

Kraut et al. [45] have done experiments that show a significant increase in the performance of users on a bicycle assembly task when they work in a collaborative work space getting video instructions from their own viewpoint from an expert compared to doing the tasks using an online figure-based manual. DuploTrack is related to this work in the sense that it provides live feedback from the user's viewpoint by tracking the model being built.

It is known that the spatial perception skills of an individual depend on many factors [26, 76, 61, 16] and while an instructional figure/video may be clear for one user, it might be confusing for another. This observation suggests that an interactive system adapting to the users' handling of the model could improve the assembly process.

Augmented Reality-based guidance. Augmented Reality (AR) techniques try to minimize the problem of perception transfer by creating an immersive environment to merge the virtual instructions and the physical model. Augmented Reality has been applied to assembly tasks and tested in user studies such as Tang et al. [78], Henderson et al. [31] and Boud et al. [14], which evaluate and demonstrate the advantages of using AR techniques over the traditional figure-based techniques. Hou et al. [33] have argued the benefits of using augmented reality and also mention the idea of playing pre-recorded animation clips at each step of the assembly which are better than static figures. In all these systems, highly specialized equipment is needed as seen in Figure 3.3b, the models are typically stationary and the motion cues are due to parallax caused by head motion. In DuploTrack, I use an inexpensive, widely available color+depth sensor along with a common computer display to provide motion cues based on the motion of a real model held in the hand.

Guidance in DuploTrack. One of the goals is to solve the problems of perception transfer and lack of control while providing the same visual cues as figures or video for structural perception. The system tracks the physical model's motion and continuously shows the replica on the screen in approximately the same orientation as the user's viewpoint with the instruction step superimposed on it. The continuous rendering on the screen can be seen as a video that is being generated on the fly from the user's viewpoint along with the instruction. This instruction mode overcomes the problems of lack of control and perception transfer.

Since the rendering on the screen is governed by the user's handling of the physical model, the user is in complete control of the pose of the model in which the instruction is being shown to him. This minimizes the need for perception transfer. The system also provides all the depth perception cues provided by the static images and the recorded video.

I experimentally compare our system with the figure-based guidance method that is most commonly used. I do not compare with the recorded video-based systems because, in essence, DuploTrack is also a video-based guidance system where the video is not pre-recorded but generated on-the-fly based on how the user views the physical model.

3.2.2 Tracking the Assembly

There has also been work to track and evaluate the correctness of assembly steps presented to the user. Molineros et al. [54] put encoded markers on each part for tracking it and detecting connections with other parts. They also precompute a connection graph between parts and feature descriptors for all configurations. Ju et al. [39] presented a system called Origami Desk which uses special hardware built into paper to sense folds and hence detect completion of predefined steps. These frameworks can be extended to a *free* mode where the user is allowed to connect any part anywhere or make a fold anywhere. However, this involves precomputation to learn descriptors for the complete space of allowed manipulations over all the parts. Searching over this space in realtime will be even harder. In comparison, DuploTrack can track and evaluate an assembly process of Duplo® blocks.

Recently, there has also been work on tracking manipulations done by user in the *free* mode. Jota et al. [38] present a system which captures and projects virtual replicas of physical objects put together by the user. However, the quality of the virtual replicas suffers due to the inherent noise in Kinect®'s depth sensing. There is no inference to establish connections between the parts. Anderson et al. [7] use special circuitry-augmented blocks to determine the assembly process of a model and then convert it to a more detailed and less blocky virtual model. DuploTrack understands the model assembly without any special hardware and also focuses on guiding the model's re-creation in a sequential manner.

In a contemporaneous work, Miller et al. [52] solve a similar problem of tracking and inferring how a Duplo® block model is built. They assume that the model always stays with its base on the table to reduce the tracking to 3 degrees of freedom (DOF): two for in-plane translation and one for in-plane rotation. Their representation is voxel-centric, rather than part-centric, requiring all voxels occupied by the model to be seen to correctly model the assembly. These restrictions limit their system to simple block models, usually built layer by layer. Their user study to measure model acquisition accuracy shows scanning errors due to tracking misalignment, hand pixels, and parts that are less visible. In comparison, DuploTrack uses 6-DOF tracking, allowing in-hand manipulation, and belief accumulation from multiple views for inference of whole-part additions/subtractions, and subsequently avoids many errors inherent in their system. Further, I enable two-way feedback between the user and the system compared to only system-to-user feedback in their work. I believe

this leads to richer human-computer interaction experience. Additionally, I report on a user study comparing motion-tracked guidance to traditional methods.

3.3 System Overview

The system is set in a playspace environment where the user works with the model on a table surface while a Kinect® depth+color camera looks down obliquely on it, as shown in Figure 3.1. At the back of the table is a display screen facing the user; when the physical model is being tracked, the display screen shows a virtual replica of the model from the point of view of the user. A set of 2×4 and 2×2 Duplo® blocks sits off to the side of the table within reach of the user. The table surface has four demarcated regions - *Play Area*, where the user builds the model, and three *Control Boxes* – *Add Box*, *Remove Box* and *Recheck Box*. The user can set up this playspace easily as described in Chapter 2. The relative layout of the three *Control Boxes* must be as shown in Figure 3.1. This is required because of the way the underlying algorithm processes data in these Boxes and I explain this in Section 3.5.

As described above, the virtual model needs to be shown on the display screen from the user's viewpoint. However, there is no mechanism in the playspace to detect where the user is after he sets up the playspace. Hence the system allows the user to adjust the view of the rendered virtual model at any point of time using a standard virtual trackball interface [32]. I assume that the user does not move around much while using the system. If he moves significantly, then the view of the display screen will need to be adjusted again manually. This is currently a limitation of the system.

The system operates in two modes - Authoring and Guidance. In the Authoring mode, the user builds a model by adding or removing blocks one at a time. The user can freely move around the model in the *Play Area* during the whole process. A tracked virtual replica is shown on the screen. To add a new block, he first places the new block in the *Add Box* and then adds it to the model. The system checks where the block has been added. Once the system detects the block's most likely position, it shows the update in blinking mode superimposed on the virtual replica in the display. If the detection is correct, the user can move to the next step directly. Otherwise, the user puts his hand in the *Recheck Box* and the system checks again for the update. To remove a block, the user removes it from the model and places it in the *Remove Box* and the system again starts the cycle of

update detection.

In the Guidance mode, there is a pre-loaded model and a sequence of block additions required to build it. As before, a tracked virtual replica of the model in the *Play Area* is shown on the screen, but with the block to be added next also shown in blinking mode superimposed on the replica. To add the block, the user first places the new block in the *Add Box* and then adds the block to the partial assembly. The system verifies the update. If the update is correct, it loads the next instruction. Otherwise, a notification is displayed providing feedback about the mistake and asks the user to correct it.

For tracking the model and inferring updates, the system needs an internal representation for the block model. I now describe that representation, followed by the system's processing pipeline.

3.3.1 Representation for Block Models

It is assumed that the model resides in a voxelized space where each voxel is of size 16mm $\times$ 16mm $\times$ 19.2mm, the official size of a 1×1 Duplo® block. The model is made of 2×4 and 2×2 Duplo® blocks either red, green, blue or yellow in color.

The following structures are maintained for the model at any point -

- List of blocks where each block has a color (red/green/blue/yellow), type (2×2 or 2×4) and list of voxels it occupies.
- Map of the complete voxelized space where each voxel is either unoccupied or maps to the block occupying it.
- Mesh model used in rendering.
- Point cloud of the model, denoted as P_v, used for 3D alignments.

The system also has a mesh model and point cloud representation for one 2×4 block and one 2×2 block which can be added to or removed from the virtual model in any color. The mesh models are from Google's 3D warehouse, converted into a dense point cloud using *MeshLab*. The point cloud P_v is a union of translated and rotated copies of these block point cloud, with points removed in areas that are covered by other blocks.

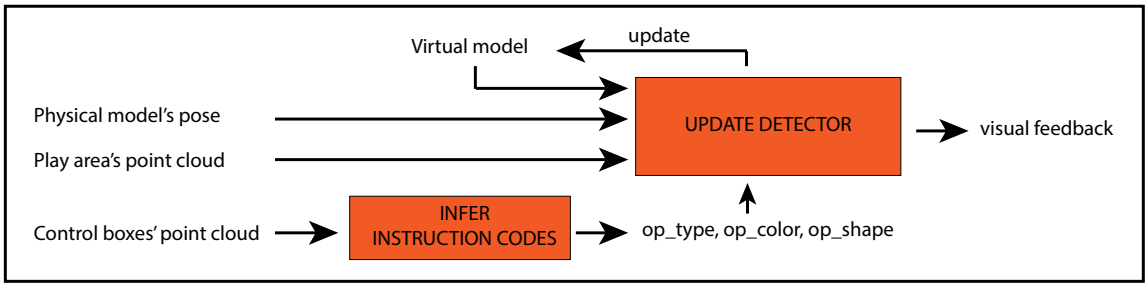


Figure 3.4: Processing pipeline: The playspace algorithms (Figure 2.1) provide the application, DuploTrack in this case, with the physical model's pose and the point clouds corresponding to the *Play Area* and the *Control Boxes*. First, the system infers the instructions codes from the *Control Boxes'* point cloud. These codes and the other data are passed to the Update Detector module which then checks for any updates to the model and accordingly updates the virtual model (if needed) and provides visual feedback on the screen.

3.3.2 Processing Pipeline

Figure 3.4 shows the work flow of the system. The Kinect® camera provides a continuous stream of images with an RGB color and depth at each pixel. The tracking and segmentation algorithms of playspaces, as described in Chapter 2, extract the 3D pose of the current model in *Play Area*. The model tracking does not work very well if its size is small. Hence I assume that the model is static when it is less than five blocks in size. I explain the effect of this assumption on the system's usage in Section 3.4.

The playspace algorithms also give out two sets of 3D point clouds –

- From the *Play Area*. This cloud is the observed current model after removing the background and user's hands.

- From the *Control Boxes*. These are point clouds from the *Add Box*, *Remove Box* and *Recheck Box*. They correspond to user's hands or any blocks that lie in these boxes and exclude the background surface.

The system first converts the point clouds from the *Control Boxes* into a set instruction codes – an *op_type* (add, remove or recheck), *op_shape* (2×2 or 2×4) and *op_color* (red, green, blur or yellow). I explain this inference in Section 3.5. The 3D pose, point cloud from the *Play Area*

and the instruction codes, are then passed on to the *Update Detector* module. This module makes any decisions about updates to the current model and provides appropriate visual feedback on the display screen. I describe the details of this module in Section 3.6.

3.4 Tracking the Model in the Play Area

The tracking algorithm, as described in Chapter 2 runs at real-time frame rates. However, the model tracking does not work well for small models below 4-5 blocks. There are two reasons for this:

- **Noise in the camera data.** The data from the Kinect® is particularly noisy near depth discontinuities. For smaller point clouds this noise is a significant fraction of the data and the tracking runs into problems.

- **Structure mismatch.** If the user adds or removes a block, we still track it using a pre-updated virtual model until the update is detected and incorporated into the model. When the model is small, the structural change of even one block confounds the tracking as the outliers from the newly added block overwhelm the points from before the update.

To start a new model the user needs to place the first block in a fixed position in the *Play Area* and add blocks to it at that location. This greatly reduces the possible candidates to search over. Once the model has 5 blocks, this restriction is removed and the user can freely move the model around. The system now dynamically tracks it as described earlier.

3.5 Inferring the Instruction Codes from Control Boxes

Each of the *Control Boxes* has a 3D point cloud associated with it, denoted by $P_{recheck}$ (*Recheck Box*), P_{add} (*Add Box*) and P_{remove} (*Remove Box*. From observing these clouds, the system needs to infer the type of operation (*op_type*), and the properties of the blocks being added – shape (*op_shape*) and color (*op_color*).

The user uses the *Recheck Box* by placing his hand in it. He uses the other two boxes by holding the block being added or removed in the corresponding box. Hence, the method to detect which operation is being done can just see which of the boxes is occupied and set the instruction codes accordingly. However, this simple maximum-occupancy based approach is incorrect. This is

because the *Recheck Box* is above the other two boxes and hence those two may also be occupied by user's arms when he rechecks. Thus, *op_type* is directly assigned to *recheck* if $P_{recheck}$ is sufficiently big (more than 10 points). This threshold of 10 points is empirically chosen and is used further in this section as a test for a point cloud being sufficiently big, and in turn the box being occupied. The other two codes (*op_shape* and *op_color*) do not matter in this case and are set to default values.

If the *Recheck Box* is not occupied then we need to check if any of the other two boxes are in use. If none of the other two boxes are occupied, then the instruction codes are set to invalid values. Otherwise, we set *op_type* to *add* or *remove* based on whether P_{add} or P_{remove} is bigger is size. Now we need to decide the block properties and set those in *op_shape* and *op_color* based on the corresponding 3D point cloud.

The block to be added or removed can only have one of the four colors – red, green, blue or yellow. Hence the colors of the points are quantized in these four categories using a simple nearest-neighbor-based color classifier. *op_color* is set to the majority color. Next, the system learns a set of eight two-way classifiers, to identify *op_shape*, one each for the combinations of the four colors and two operations – add or remove. The classifier is a threshold value over the size of the point cloud. The threshold value for an operation and a color is computed by averaging the two observed point cloud sizes when a 2×2 block and a 2×4 block of that color are placed in that operation's box. If the observed size is greater than the threshold, *op_type* is set to 2×4 and 2×2 otherwise.

We need eight classifiers because the observed point cloud sizes vary with the block color and the operation. The dependence on block's color is because different shades of the four colors get segmented out differently by the skin-segmentation algorithm. The dependence on operation is because the cloud size varies with the position of *Control Boxes*.

3.5.1 Limitations

This classifier for the block's shape is very naive and has limitations. It does not handle occlusions. For example a partially occluded 2×4 block can be classified as 2×2 because the latter is a subset of the former. This classifier may not work when we introduce more block shapes in our system. Hence we may want to use a more sophisticated feature, for example the number of studs. This new classifier should work well under the assumption that the blocks are made up by putting together

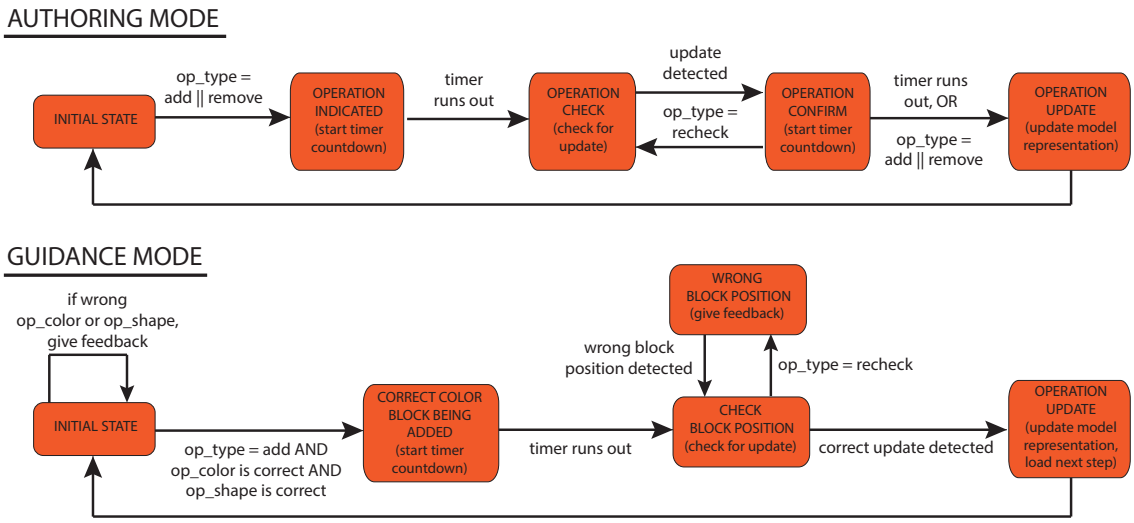


Figure 3.5: The Update Detector is implemented as a finite state machine with user input and tracked data as input to update the model. In Authoring mode, it detects an update and applies it unless a recheck is requested by the user. In Guidance mode, it detects and performs the update and displays feedback to the user if there is a mistake.

1×1 basic Duplo® units and are held with the studs facing up and completely visible. Handling curved and articulated parts is a bigger challenge which I would like to address in future.

3.5.2 Making the inference more robust

The system infers the instruction codes for every frame and passes them to the *Update Detector*. As the user's hands enter or leave the *Control Boxes*, the occupancy and the observed clouds will vary rapidly and per-frame inference may vary with them. Thus, we need to find a way to stabilize this potentially noisy inference. I design a simple stabilization approach for this.

Before passing a valid set of instruction codes as output, the system waits until that inference has stabilized over a window of 15 frames, i.e. for about half a second. Until then, the system sends invalid codes as outputs. Also, once a valid inference is passed as output, the system does not process a subsequent time window of 60 frames, i.e., a time frame of two seconds. The window sizes are empirically chosen based on the typical speeds at which users worked with the system. These policies are observed to provide robust inference of instruction codes.

3.6 Detecting Model Updates

The *Update Detector* module gets a continuous stream of inputs in addition to the point cloud of the "un-updated" virtual model, (P_v) –

- Point cloud of the physical model in the *Play Area*, (P_c).

- The tracked pose of the physical model, (T). The pose T can also be seen as a transformation that aligns P_v, which lies at the origin, to P_c. If a block is added or removed the alignment may not be perfect. The system also extracts the points from P_c that do not find a good match in P_v and mark them as outliers. When a new block is added, these outlier points indicate the possible location to search for the addition.

- The instruction codes – type of operation (*op_type* – add, remove, recheck) and color (*op_type* – red, green, blue, yellow) and shape of the block that is added or removed (*op_shape* – 2×2, 2×4).

Separate finite state machines implement the Authoring and Guidance modes as shown in Figure 3.5. These are a formal representation of the system usage already described in detail in Section 3.3.

In Authoring mode, when the user indicates an operation, the system checks for changes in the model as the user moves the model around in front of the sensor. Once the system detects the update, it echoes this in the display. If the user thinks the system is in error and asks for a recheck, the system rechecks; otherwise, it updates the model and moves to the next step.

In Guidance mode, the system checks for the correct color and position of the block and gives appropriate feedback to the user. It goes to the next step automatically if the detected update matches the instruction otherwise it waits for the user to correct the mistake and ask for recheck. Note that only block additions take place in the Guidance mode. Any block removals during the Authoring mode effectively undo the previous addition of that block.

3.6.1 Creating the Candidate Set for Updates

At any stage in the assembly the system maintains a set of candidate updates to the model. The candidate updates are one of two types, addition or removal, depending on the indicated operation in *op_type*. In addition mode, the voxel space is traversed to find sets of unoccupied voxels where a new block could be added. The shape of the block to check for comes from *op_shape*. To ensure connectivity in the model, these candidates must be connected to at least two occupied voxels; i.e., I assume the user is creating a single, rigid model, each piece connected to the model using at least two studs.

In removal mode, the candidates are simply any block that can be removed without leaving the remaining model disconnected, i.e. (1) either the space directly above those blocks or below is completely free and (2) they are not the only connected neighbors to any other blocks connected to them. The second condition does not apply when the remainder model has just one block. This analysis is done via the voxelized representation of the model.

One more candidate is added to both modes which corresponds to *no addition/removal*, the state of the model just before the user adds/removes a block.

For each valid candidate, j, the system maintain a belief, $b(j)$, which denotes the belief that candidate j should be chosen to update the model. The current belief is based upon the input stream consisting of the physical model's point cloud P_c and the transformation T that aligns P_v to P_c.

3.6.2 Updating the Belief Distribution

After each new block addition or removal, all the $b(j)$ are set to 0. The $b(j)$'s are updated each time new camera data comes in. It is often difficult to determine which block has been added or removed from a single viewpoint due to input noise, occlusion, and structural ambiguities. Thus, the system accumulate beliefs from more than one pose of the model as the user turns the model to reveal new views.

From each pose, the possible candidates are scored based on how well the camera point cloud P_c, matches with the virtual point cloud corresponding to candidate j, denoted by P_j. For an addition candidate, P_j is obtained by adding the (colored) point cloud of the new block to P_v and removing the points that get hidden by this block. For a removal candidate, P_j is obtained by removing the

point cloud of the block from P_v and adding back the points that were hidden by this block. For the no addition/removal candidate, P_j is simply equal to P_v.

To compute the matching score, first P_c is aligned with each of the P_j's using ICP. The P_j's are structurally close to P_v and P_v has already been aligned with P_c using the transformation T. Hence T is used as the initial transform for these ICP [12] alignments. Then the match error for each candidate is computed using the same matching metric as used for comparing poses for tracking in Section 2.3.2. For addition candidates, the error is set to infinite if the majority of outliers which occupy the voxels of the candidate are not of the indicated color *op_color*. The same is done for removal candidates if the corresponding block is not of the indicated color.

The match errors are sorted in increasing order. The top three ranking candidates get the belief scores of 3, 2 and 1 and the rest get a score of 0. To select a best update candidate, the system requires beliefs from at least three poses separated by at least 10 degrees of rotation to make a decision (for small models, when the system is working with a fixed pose, this requirement is removed). To accumulate beliefs from three well-separated views, the system begins with the initial transformation, call this T_1, and average beliefs into a belief set $b_1(j)$ over subsequent nearby observations until T has changed by at least 10 degrees of rotation from T_1. This establishes a second transformation, T_2, with a new set of beliefs $b_2(j)$ for each candidate. The system continue to collect beliefs over subsequent nearby observations, averaging them with the beliefs corresponding to the nearer of T_1 or T_2. We also continuously check for a new transformation, T_3, at least 10 degrees from both T_1 and T_2, and, if found, begin a new average set of beliefs $b_3(j)$. More views and distributions are added if the user continues moving the model to orientations that are at least 10 degrees away from all previous views.

Once belief sets associated with at least three well-separated views have been accumulated, each set is normalized to sum to 1 and then all normalized sets are summed to get an overall set, $B(j)$. If the ratio of the highest scoring candidate in this set is at least 1.25 (chosen empirically) times that of the second best candidate, then the highest scoring candidate is declared as the winner. If a winner cannot be declared, then the system waits for more camera data from new poses to update the beliefs, continuously checking for a winner as beliefs are updated. If the *no addition/removal* candidate is the winner, which can occur if the user has not yet added/removed a block, the system resets the belief sets and re-starts checking for an update.

3.6.3 Moving to the Next Step

If the system selects a candidate as the update but the user then asks for a recheck, that candidate is removed from further consideration, reset the belief sets, and the update checking process restarts. In practice, I find that the user needs to request a recheck less than one out of twenty additions or removals.

Once the model update has been identified (and approved by the user in Authoring mode), the internal representation of the model is updated. The model's virtual cloud P_v is set to the point cloud corresponding to the winning candidate. For block addition, the new block is appended to the list of blocks and map the corresponding voxels in the voxel space to it. For block removal, it is removed from the list and the corresponding voxels are marked as unoccupied.

3.7 Performance and Applications

The system runs in realtime on a desktop PC with 12-core, 3.33GHz Xeon CPU and uses at most 500MB of RAM. To achieve this performance, the implementation is highly multithreaded with separate threads for tracking, rendering and checking updates. Within the update thread, the candidates are evaluated in parallel.

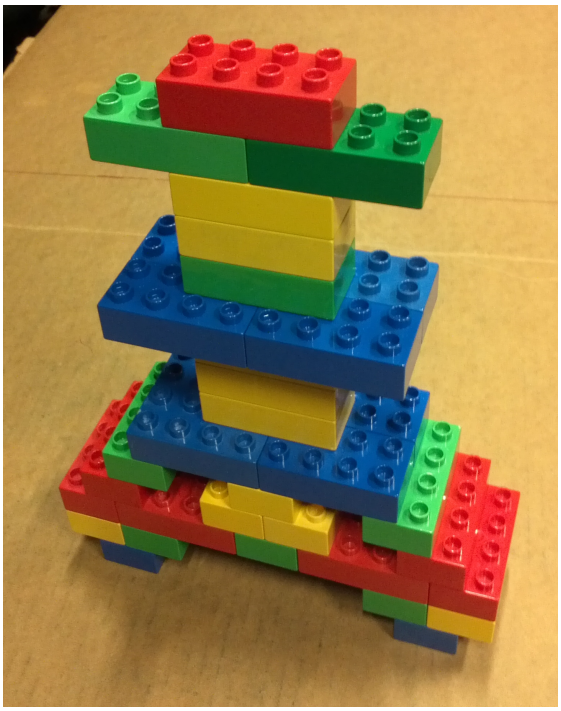
The system takes about 2-5 seconds to infer each model update. The tracking works in realtime although it lags slightly if the model motion is fast. I have used the system to build models up to 85 blocks in size. At that point the tracking speed reduces to about 5 frames per second. Figure 3.6 shows a few models that users have authored while using DuploTrack.

The bottleneck of the system is nearest neighbor correspondence search in ICP alignments for tracking and candidate evaluations. These searches can be done in parallel for 3D points in a cloud, but the number of CPU threads is limited. In the future I believe that my system can benefit from exploiting the parallelization potential in GPUs as shown by recent works like *KinectFusion* [35].

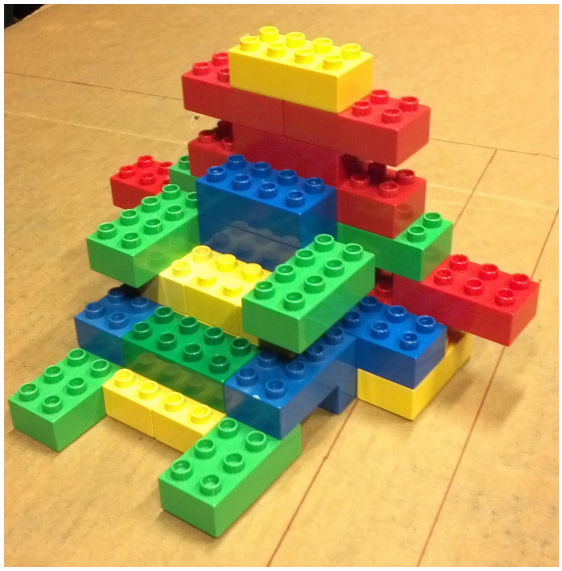
I now describe a few applications of our system.

3.7.1 Generating Tutorials for Model Assembly

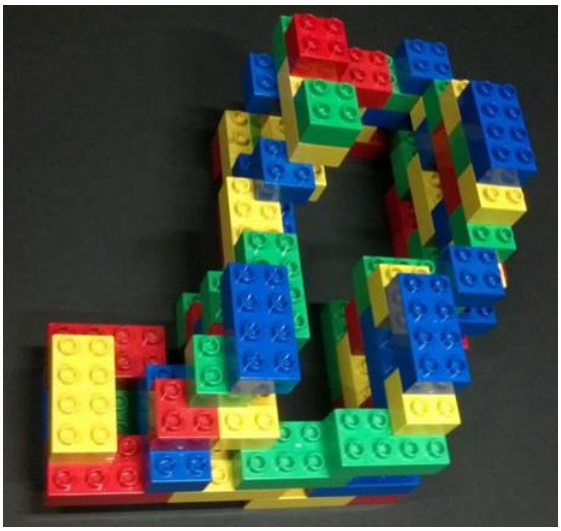
The captured assembly of a model is a sequence of add or remove operations. To generate an assembly tutorial, we would like to omit any steps that involve adding and then later removing a



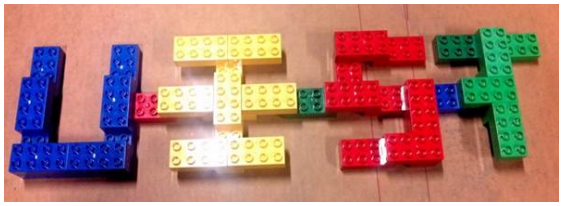
(a)



(b)



(c)



(d)

Figure 3.6: Models authored using our system.

block. I delete such add/remove operations from the representation. This may cause the remainder of the sequence to have block additions which do not connect to any blocks added before them. I reorder the steps by postponing the addition of such blocks until they have their first connection with the prior model. I describe and analyze the reordering algorithm in detail in Appendix A. This representation can now be used to then generate tutorials or explanations in form of images with or without annotations [49, 53].

3.7.2 Ease of Sharing and Recreating Models

The captured assembly sequences from the Authoring mode can also be used as input to our Guidance mode which uses exactly the same hardware setup. This allows for an easier way for people to create and share their models with other users or save these representations for future re-creation.

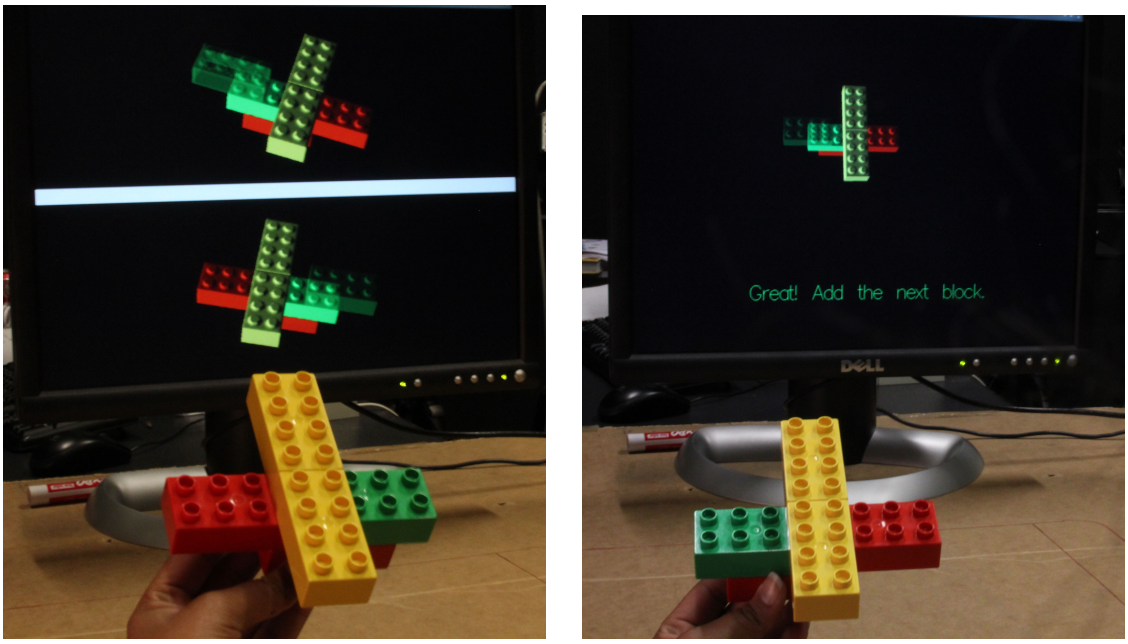
3.7.3 Access to Virtual Replicas

The output of the Authoring mode is a virtual 3D replica of the model. These virtual models could be used for different purposes, e.g., as content for games and animations.

3.8 User Study

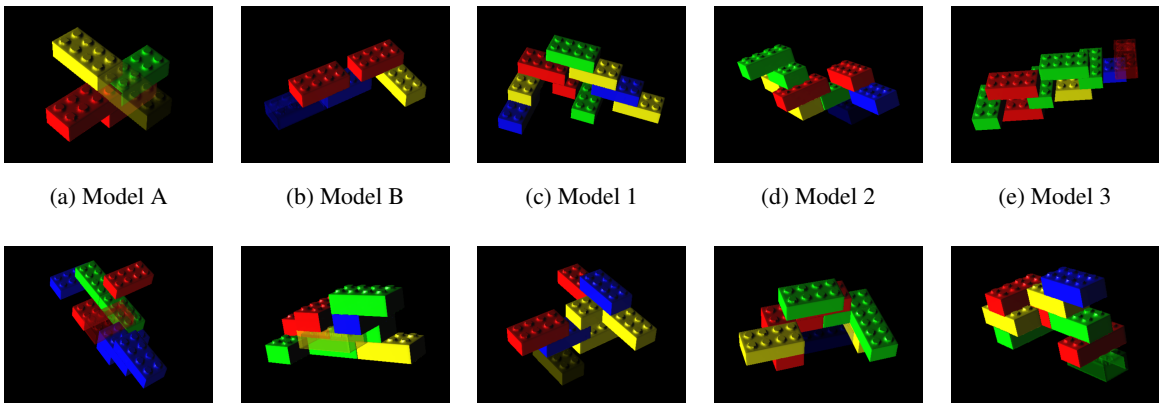
There are many aspects of the Authoring and Guidance system for which we can ask questions that can best be answered by observing user behavior. In this work, I conducted a study that focuses on the Guidance system as it can be compared to other more traditional guidance modes.

In particular, I conducted a user study to focus on the differences between two interfaces for providing instructions for adding blocks to Duplo® models. The first interface (*Baseline*) provides two static views of the model (with the new block) on the screen. The new block blinks, alternating between opaque and semi-transparent. These views were manually chosen to give the best possible view of the new block from two different directions. Figure 3.7a shows a photograph of this system in progress. This interface is close to the current methods found in user manuals for the assembly tasks. The second interface, (*Track*), is my tracking-based system which shows the virtual model on the screen in the same pose as the physical model in the user's hands. The display updates in real-time as the user moves around the model. The block to be added is again shown in blinking



(a) *Baseline* interface (b) *Track* interface

Figure 3.7: Photographs of the two guidance interfaces that I test in the user study. The *Baseline* interface depicts two static views of the model. In the *Track* interface the orientation of the model tracks that of the model in the user's hand.



(a) Model A (b) Model B (c) Model 1 (d) Model 2 (e) Model 3

(f) Model 4 (g) Model 5 (h) Model 6 (i) Model 7 (j) Model 8

Figure 3.8: Initial models used in the user study tasks. Models A and B have 4 blocks each and remaining models have eight blocks each. Participants added one block to each of the 8-block models and 12 blocks in sequence to each of the 4-block models. The transparent block in each of the models is the block that needs to be added next.

mode as in *Baseline*. Figure 3.7b shows a screenshot of this interface. Please refer to supplementary videos which show how participants use these interfaces.

For *Baseline*, I decided against using the more traditional way of showing before and after figures of the step. Rather, in the two interfaces being tested, I use the same visualization for the new block to be added to avoid the results being confounded by this difference. Also, I do not use the mistake-detection capability in *Track* since I wish to directly compare the instruction modes.

I conduct two tests, one in which I focus on single-block additions, and the other on multiple sequential-block additions.

3.8.1 Task Design

I assembled ten initial block models, shown in Figure 3.8. Models A and B initially have four blocks each and Models $1-8$ have eight blocks each. The task in the study was to add one block to each of the 8-block models, and 12 blocks sequentially to the two 4-block models. Each participant had to complete half of the tasks (4 one-block additions and one 12-block addition) with one of the conditions, and then the remaining tasks with the other condition. I shuffled the models randomly between the two conditions for each participant. Also, the order in which the participant used the conditions was randomly decided to counter the effect of any 3D-perception learning that might occur by completing the tasks.

For each block addition, I measured the time taken to add that block. I also noted if the block addition was correct or not. I analyze the effect of the interface conditions on these two dependent variables - the time taken to make the update and correctness.

3.8.2 Procedure

Before starting the assembly tasks, the participants were asked to answer a set of ten questions to test their spatial visual ability. These questions had a mixture of Mental Rotation questions [72] and 3D structure assembly and paper-folding visualization questions, which are commonly part of a spatial IQ test. We might expect that the people with high spatial visual ability perform well in any interface conditions and hence I wanted to test this possible dependence. I also collected information about the gender, education level (undergraduate or graduate) and previous experience with Lego® blocks

on a Likert scale of 1 to 5. In my analysis, I wanted to analyze the possible effects of these factors also.

After completing the initial set of questions, the participants started using one of the two interface conditions. Before starting each condition, I gave them a demo of the interface with a training model, separate from those in the tasks, and asked them to practice adding 4 blocks to it sequentially. Under each condition, they first did the single-block additions to four of the 8-block models, and then completed twelve block additions sequentially to one of the 4-block models. After completing every block addition, the participants were provided feedback about the correctness of the block addition. Providing feedback was important for the case of the 4-block models because I did not want the effect of a mistake at an earlier stage to cause a mistake at a later stage. I did not record the time taken for corrective feedback and correction in our time record of the task completion.

After the participants finished all the tasks, they were asked to fill a post-study survey which asked for qualitative feedback about the two conditions. I asked them about their preference for the systems, if any, and also for any other interface ideas that could help guide them better. I report qualitative comments.

3.8.3 Participants

Sixteen participants (eight female, eight male, ages 20 to 30) volunteered. Twelve of them had built models with blocks as a child and all of them had experience in doing some 3D assembly tasks like furniture, electronics etc. Each participant's study lasted for about 45 minutes.

3.8.4 Results

I analyze the results of the experiment in three ways. First, I discuss the one-block additions to eight of the ten models across the users and the conditions to make a quantitative comparison of the two interfaces. Second, I use the measurements from the twelve sequential block additions to the 4-block models to see how the time taken varies when the basic model remains the same. I analyze each of the two models separately for this. Third and last, I report the qualitative feedback from our participants about the two interfaces.

Comparing the Interfaces for the One Block Additions

I analyze the results using two dependent measures - the time taken to complete the block additions in milliseconds and the correctness.

I perform a mixed-model analysis of variance where *Time* is the dependent factor, and *Participant* and *Model* are random effects. Modeling *Participant* as random effect accounts for any variation in individual performance, and modeling *Model* accounts for any difference in difficulty of block addition across the models. I use the following variables as fixed effects -

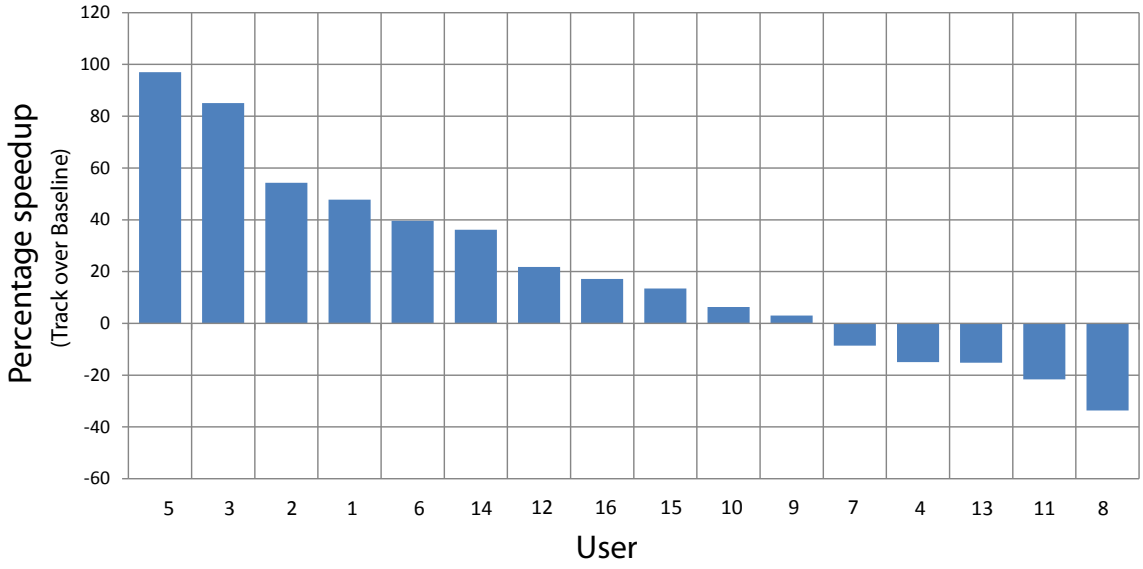
- *Gender*
- *Score*: The score of each participant on the set of ten spatial IQ questions.
- *Education*: Level of education - undergraduate or graduate.
- *Experience*: Personal experience with building Lego® models (increasing scale of 1 to 5).
- *Interface*: The interface condition - *Baseline* or *Track*.

The mixed-model analysis reveals that only *Interface* has a statistically significant effect on *Time* ($F(1,104) = 4.4932$, $p < 0.05$). The average time taken for the tasks using *Baseline* is 21.809 seconds (stdev = 10.1s) and for the tasks using *Track* is 18.871 seconds (stdev = 8.1s), an improvement of about 14%. The 95% confidence interval for the difference in the mean times is from 0.189 seconds to 5.687 seconds of improvement for *Track* over *Baseline*.

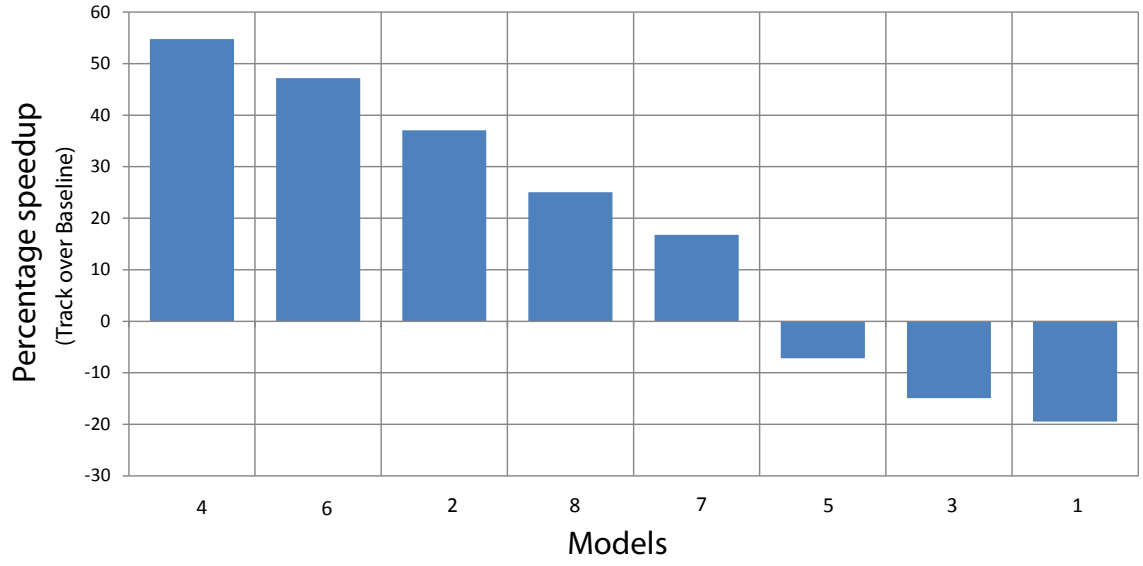
I define the *speedup* as the percentage increase in the speed of performing the step using *Track* vs *Baseline*. Specifically,

$$speedup = 100 * (\frac{Time(Baseline)}{Time(Track)} - 1)$$

A value above zero indicates that *Track* took less time and vice versa. In Figures 3.9a and 3.9b I show the *speedup* for individual users and models respectively. Although the per-user and per-model time averages have been aggregated over a small sample of the models and users respectively,

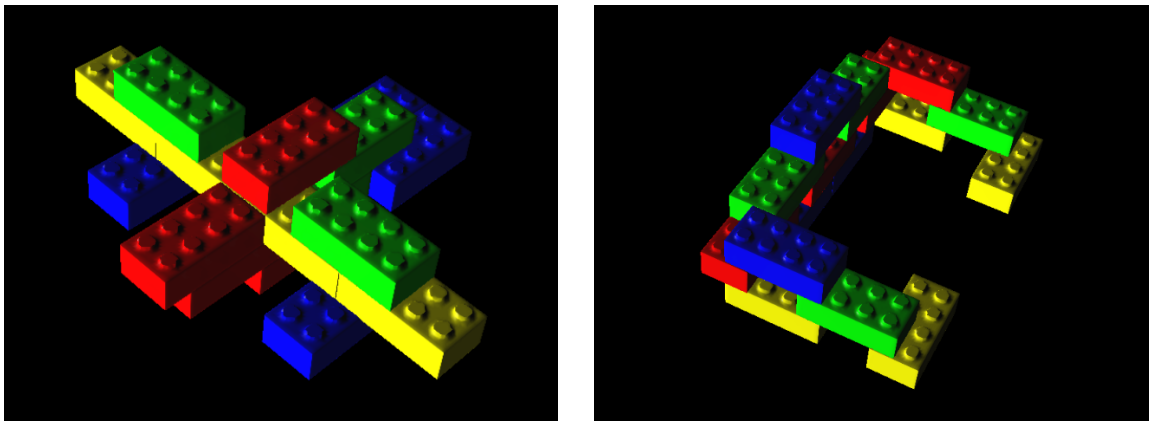


(a) Percentage speedup for users.



(b) Percentage speedup for models.

Figure 3.9: Percentage speedup over all the one-block addition tasks (value greater than 0 means that the *Track* interface takes less time).



(a) Model A. (b) Model B.

Figure 3.10: The two 16-block models, each built by adding 12 blocks sequentially by the participants.

they provide evidence for the statistically significant improvement from the mixed-model variance analysis.

In a second point of comparison between the interfaces, the users made 3 mistakes out of total 64 single-block additions while using *Baseline* while no mistakes were made in the same block additions while using *Track*. I observed that the participants preferred to take longer times to complete a step rather than making a mistake as they were correcting their actions continuously before actually adding the block to avoid making the mistake.

Comparing the Interfaces for Multi-Block Additions

Here, I analyze the times to add blocks one after the other to the same model. I want to see if there is any dependence on the times taken as the model grows, or any dependence on the particular steps or on the interface. For this, I consider two 4-block models to which the participants make 12 sequential block additions using the two interfaces, one interface per model. I do a mixed-model analysis of variance for each of the models separately. Figure 3.10 shows the two complete models and I refer to them as Model A and Model B. As before, I use *Participant* as a random effect and *Interface* as a fixed effect. I add the size of the model (number of blocks) at every step, denoted by

variable *Step*, as another fixed effect. In this analysis, I will call a step harder if it takes a longer time to add that block.

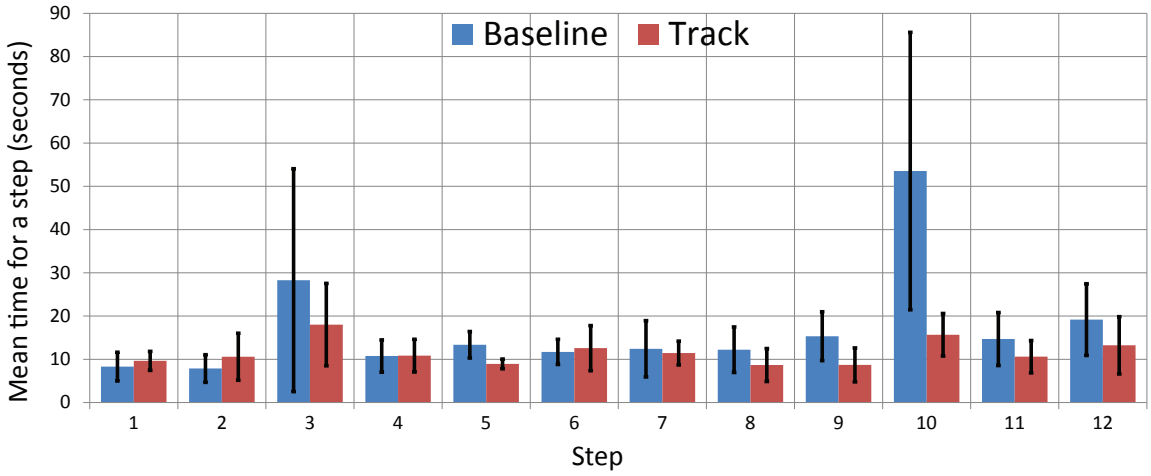
For Model A, the mixed-model analysis reveals that *Time* is significantly affected by both *Interface* ($F(1,11) = 5.3956$, $p < 0.05$) and *Step* ($F(11,154) = 11.6386$, $p < 0.0001$). The mean times for *Track* and *Baseline* interfaces over all the steps are 11.577 seconds (stddev = 5.38s) and 17.303 seconds (stddev = 17.08s) respectively. The 95% confidence interval for the difference in the mean times is from 0.438 seconds to 11.014 seconds of improvement for *Track* over *Baseline*. Figure 3.11a shows the mean times for the individual steps. We do not observe any correlation between the step (which is also the number of blocks in the model) and the times taken. We do observe that some steps take longer to complete than others, and hence are harder. The *Interface* by *Step* interaction is significant ($F(11,154) = 6.1956$, $p < 0.0001$) with *Track* showing a stronger effect on harder steps. Figure 3.11b shows the percentage speedup for different steps with the statistically significant data points marked in red. By observing the significant data points, we can infer that using *Track* can help users to understand the structure better particularly when the block update step is harder.

The participants made 7 mistakes in building Model A using *Baseline* compared to none using *Track*.

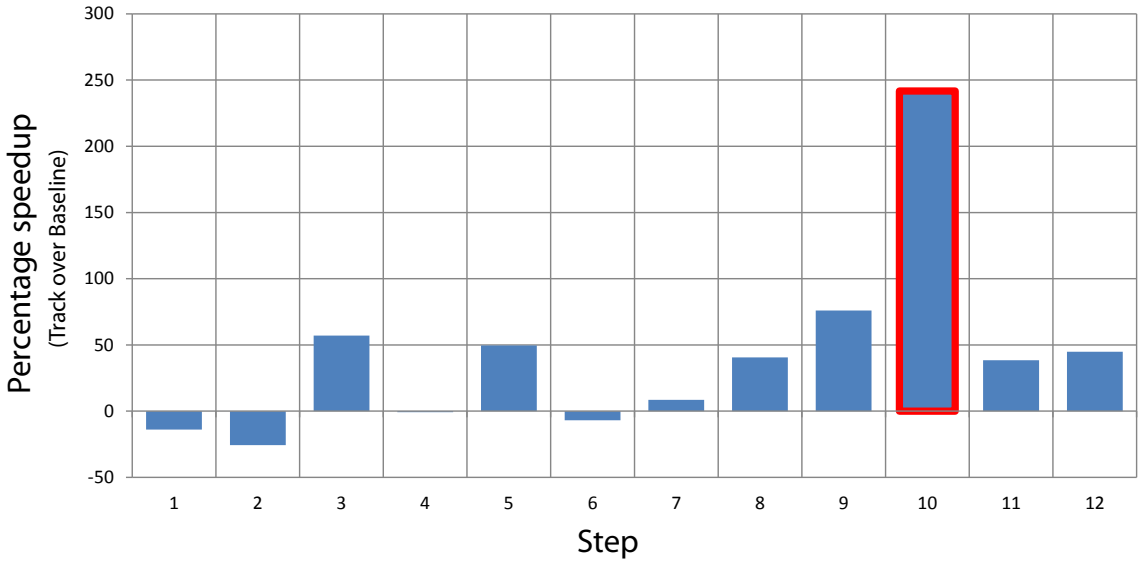
For Model B, the mixed-model analysis reveals no significant effect of *Interface* on *Time*. The mean times for *Track* and *Baseline* interfaces over all the steps are 10.028 seconds and 10.224 seconds respectively. *Step* does have a significant effect ($F(11,165) = 3.5202$, $p < 0.0002$) which indicates the varying difficulty level across steps. Figure 3.12a shows the mean times for each step. The *Interface* by *Step* interaction does show significant improvement of *Track* in some cases. Figure 3.12b shows the percentage speedup for this model with the statistically significant data points in red again supporting the observation that using *Track* can help the harder steps.

The participants made no mistakes while building this model using either of the interfaces.

Perhaps the most interesting observation with the multi-step models is that *Track* provides larger improvements for the steps in the models which are harder than the others. To further quantify this observation, I plot the speedups against the mean time taken to complete a block addition using *Baseline* across both all the ten models. Higher mean time for a step indicates that it was relatively harder. Figure 3.13 shows this scatter diagram. I fit a line using least squares to the points (shown in red) indicating a positive correlation, ie. the speedup obtained using *Track* over *Baseline* increases

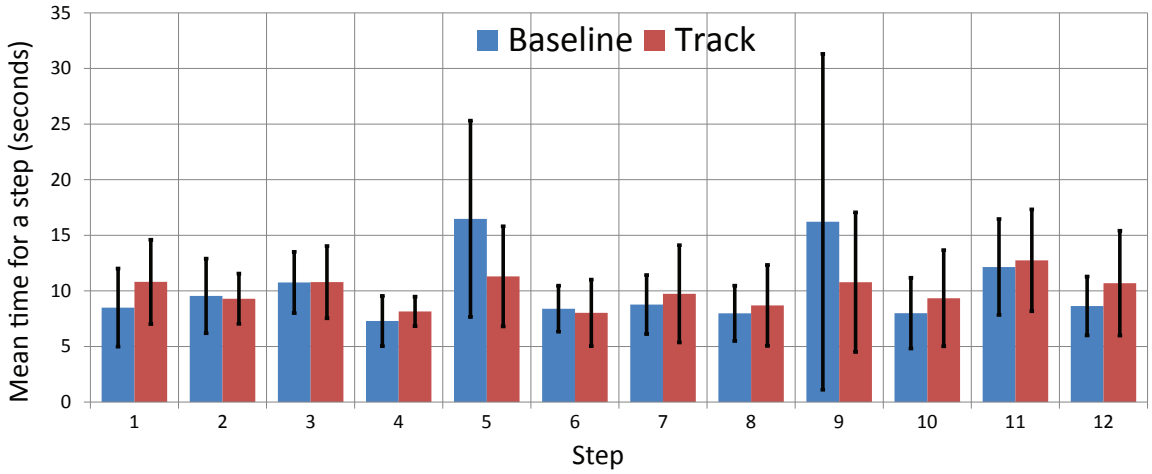


(a) Mean time taken per block addition and standard deviation (seconds). Less is better.

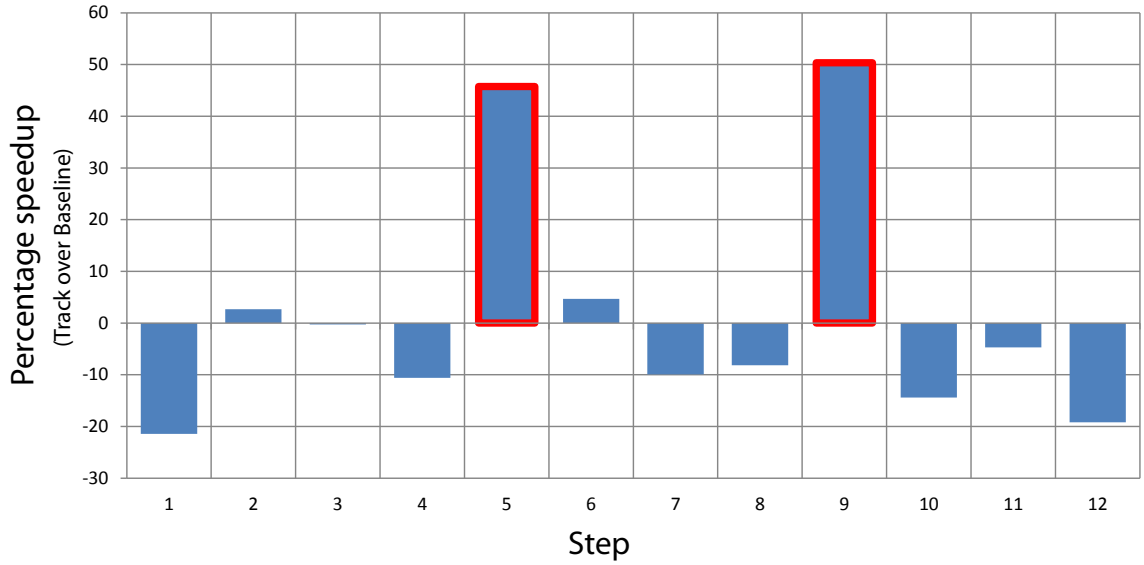


(b) Percentage speedup (value greater than 0 means that the *Track* interface takes less time. The red outline indicates statistically significant result ($p < 0.05$))

Figure 3.11: Metrics for Model A – Mean time per block addition and Percentage speedup.



(a) Mean time taken per step and standard deviation (seconds). Less is better.



(b) Percentage speedup (value greater than 0 means that the *Track* interface takes less time. The red outline indicates a statistically significant result ($p < 0.05$))

Figure 3.12: Metrics for Model B – Mean time per block addition and Percentage speedup.

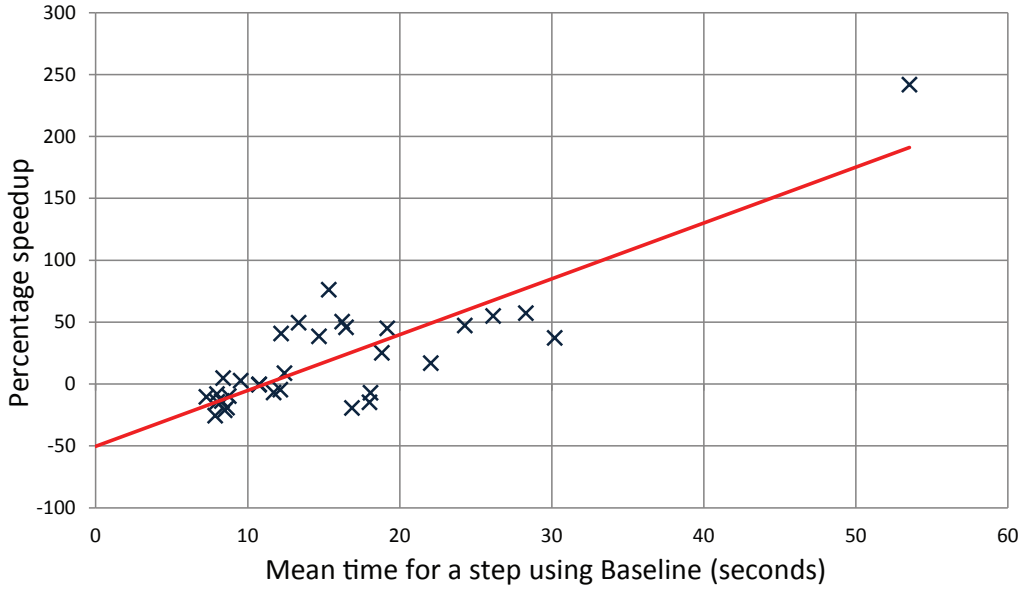


Figure 3.13: Scatter plot of percentage speedup vs mean time taken to do a block addition using *Baseline* (an indicator of the step's hardness). Fitting a line using least squares (shown in red) indicates a positive correlation between the mean time and speedup obtained by using *Track* over *Baseline*.

with the difficulty of the step.

Qualitative feedback

Eleven of the sixteen users said that they preferred *Track* since it gave them flexibility of moving the model around and understanding its structure better. It was less mentally taxing. Three users preferred *Baseline* since they wanted the guidance to remain static and preferred to move the model around to match the shown view and then add the block. Based on this, we can imagine another interface which does the tracking in the background, and allows the user to ask for the instruction in the current pose of the model if desired.

All the users said that the assembly process using *Track* was more enjoyable experience. They talked about the ability to record stories using models and building virtual models on the fly. As one user said, "This tracking-based system will make playing with Lego® blocks more fun".

Based on our interactions with the users, I feel that the type of guidance varies with users. While

the tracking-based system does show improvement in model-building times and fewer mistakes made, some users may still prefer the static views or getting static-views on demand.

Results Summary and Discussion

The results suggest that dynamically updating the pose of a virtual model to correspond with the real model in a user's hands can improve both speed and accuracy. I observed no mistakes made with the *Track* system while a total of 10 mistakes were made with the static *Baseline* instructions. I also observed significant speedups for the tasks with the dynamic systems.

Clearly, I have only scratched the surface of the questions raised. All of the models are quite simple compared to the richness of Lego® models available. I can conjecture that our results may even be stronger as the complexity increases but I have only limited evidence for this from our experiments. The simplicity of our models is also constrained by the technology. The resolution of the Kinect® device precluded using the smaller Lego® block for example. I can hope that this aspect of the system cam improve over time.

3.9 Conclusion and Future Work

I have demonstrated a system DuploTrack which tracks an evolving Duplo® block model in real-time. In Authoring mode, the system learns the assembly through block additions and removals. In Guidance mode, a user is prompted to construct a predefined model by presenting instructions in the same pose as that of the physical model. It also provides feedback about mistakes and appropriate corrections to the model. I discuss the shortcomings of existing static figures or videos of the instruction steps and show how my guidance method avoids these. A user study comparing my system with the traditional figure-based guidance method suggests that my method is able to aid users' structural perception of the model and hence leads them to make fewer mistakes and construct the models in less time.

Some people still prefer the traditional system because they do not want the instruction to move with the physical model. In the future, a system which tracks the physical model continuously but only shows the instruction in the current pose on demand may satisfy all users. Based on my informal discussions with the participants, I noticed that different people tend to use different features of

the image for adding the blocks. While some looked at edges, others counted the number of studs in the blocks, and others looked at visibility cues. An interesting future direction would be to analyze what features people tend to use for different types of assembly tasks and adapt the presentation accordingly. Additionally, I can explore new human-computer interactions for guidance like the user being guided to bring the physical model in the same pose as that of instruction. A user study could also be done to correlate people's spatial IQ scores to the guidance method that they prefer.

I hypothesized that depicting the virtual model on the screen in the same pose as the physical model minimized the need of perception transfer. To further reduce that, we can try replacing the display screen in our current system with a projector attached with the camera. The projector can project the instruction step directly on the physical model or near it on the work surface. Further, the projector can be used to display extra information about the model or its different parts for educational or guidance purposes. I discuss the various augmentations and modifications to the playspace framework in Chapter 6.

I want to extend this work in future to handle blocks of more sizes and shapes. Rigid blocks made of standard 1×1 Duplo® units are not hard to include as they easily fit into the voxelized representation. We can identify the shape of the block when the user puts in it *Add* or *Remove* box and evaluate the appropriate candidates. Developing descriptors for recognizing parts of curved shapes and articulated parts also has interesting challenges. It would also be useful to move away from a block-at-a-time update approach and allow for making sub-assemblies and merging them. This might require a different model representation and present more computational challenges.

I would also like to extend our framework for other types of assembly tasks like furniture assembly, and home repairs. This may require waiting for higher resolution sensors than the Kinect® or using multiple sensors and then fusing the data. However I also think some of the complexity can be overcome with better algorithms. Tracking and candidate evaluation algorithms can be vastly sped up by the use of GPUs in future. I discuss some more applications of the playspaces for the assembly tasks in the concluding chapter.

Chapter 4

DIGITAL STORYTELLING IN PLAYSPACES

"Life itself is the most wonderful fairytale of all." – Hans Christian Andersen

4.1 Introduction

Visual story telling through puppets has been an age old art form. In the past few decades, digital animation industry has grown leaps and bounds and has taken the concept of puppetry to an entirely new level. The digital animators and artists use software tools like Maya, 3ds Max and Blender to create high quality virtual animations. Currently these animation tools have a very complicated interface to enable high fidelity controls for the animators. However, I believe that there should exist more natural intuitive interfaces which allow novice users to easily create digital animations even if it lowers the quality of the result.

(a) Toys for story telling.

(b) 3D-Puppetry system for story telling.

Figure 4.1: Natural and intuitive interfaces for story telling. (a) Toys and puppets are the traditional ways of natural story telling. (b) The 3D-Puppetry system tracks the moving physical objects using a Kinect® camera and renders their tracked virtual replicas on the screen in realtime to create an animated story.

An intuitive interface to tell a visual story for novice users is through physical puppets and toys (Figure 4.1a). Hence we can develop systems which automatically track and transfer the acted out motion to virtual characters and hence record an animation. There has been prior work in this domain. Microsoft's Kinect® system [75] and commercial motion capture systems use human pose tracking to control a virtual avatar for games and animations. Chen et al. [17] map the tracked human skeleton to a virtual object to deform it in realtime. Johnson et al. [37] track a plush toy with embedded sensors and map motion gestures in the physical space to virtual motions of the replica. Lee et al. [47] and Dontcheva et al. [20] use marker-based tracking of fixed physical primitives to move the virtual objects attached to them. All of the above systems track specific objects and hence restrict the objects that a user may use for acting out a story.

Barnes et al. [10] present a Video-Puppetry system which allows the user to design 2D paper-cutouts as puppets and feed them in the system. The system tracks the motion of these puppets on a 2D surface in realtime and maps it exactly to the virtual replicas. Inspired by this we developed a system 3D-Puppetry [30] which allows the users to act out the story using any physical objects and the system converts the recording into an animation with virtual object replicas.

The 3D-Puppetry system uses the framework of a playspace. Figure 4.1b shows a user using the system. As is the case with playspace framework, the user first scans in the physical objects that he intends to use in the story. Then he acts out the story using objects in the *Play Area* which the system tracks in realtime and renders replicas in a pre-selected virtual environments on the display screen in front of the user. This rendering is also recorded as a video which is the resulting animation. This system allows user to use some keyboard and mouse-based controls to edit the animation later by changing light positions, camera viewpoint etc. We invited novice users to use the system and the overall feedback about the experience was very positive. The users were able to start using the system in almost no time and record simple yet creative stories. This work was led by Robin Held and Maneesh Agrawala from UC Berkeley and hence does not completely come under the scope of my thesis. I refer the readers to the 3D-Puppetry paper [30] for more details.

One significant drawback of 3D-Puppetry and all such performance-based 3D animation systems is that they require the user to discard a take if any aspect of the motion was incorrect or undesired and then repeat the take from scratch, hoping the next one will be just right. Yet, each take may contain some parts that are better than others even if no complete take is satisfactory.

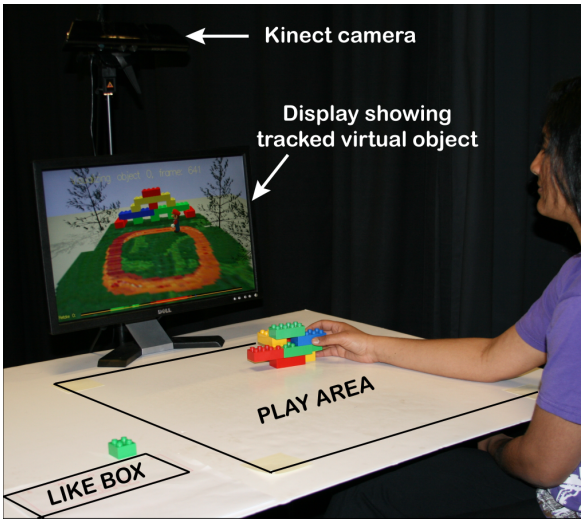


Figure 4.2: A user working with the MotionMontage system to record multiple takes of a 3D animation. The user records takes for objects, one at a time, by moving the controller in the *Play Area* while the Kinect® camera tracks it. The *Like Box* helps the user to annotate the takes which is used for merging them into a motion montage.

It is also a standard practice in movie industry where multiple takes may be required for one shot. Such re-takes allow the director to capture variations in the dialogue, and try out different positioning of the actors and camera angles. In performance-based 3D animation, a director may ask the animator to perform multiple takes in order to test different styles of motion (e.g. a more energetic performance versus a somber performance) and find the most appropriate style for the shot. In film, there are instances of a single shot requiring many takes; for example, one shot in the movie "The Shining" was reported to have needed 148 takes before the director, Stanley Kubrick, was satisfied [50].

Motivated by this, I explore the idea of working with multiple takes for performance-based 3D animation. In this chapter, I present *MotionMontage*, a system that allows novice animators to combine multiple takes into a desired result, called the *montage*. The animator works in a playspace environment and records one object's motion at a time, as shown in Figure 4.2. The takes may vary in style or trajectory of the motion. The system allows the animator to temporally annotate each take based on their like or dislike of various parts of the take. The system then merges the best parts of each annotated take into a single composite montage using a combination of dynamic time warping

and optimization of a Semi-Markov Conditional Random Field. The user can repeat the process for the same object to further refine the montage or to animate other objects.

The system also allows animators to create layered animations in which multiple objects are moving at the same time. Although the animator must perform the motion of one object at a time, the system plays back all of the motions together. To aid the animator in coordinating the motions of the different objects, we introduce spatial markers indicating the positions of previously recorded objects at user-specified points in time. The animator can place multiple spatial markers for one or more objects to help plan the motion of the current object.

The key contribution of this work is a novel technique for combining multiple takes of an object's motion into a motion montage using a simple annotation-based interface. I also introduce the use of spatial markers to aid in layering the motion of multiple objects. I report on a formal user study that validates that the animation montage is generally perceived to be better than any individual take. I evaluate this from both the perspective of the users who created the animations and from others who only view the animations. In addition, we report qualitative feedback from a short informal study of the use of markers to record multi-object animations.

Organization of the chapter.

4.2 Related Work

My system for combining multiple takes of performance-based 3D animations builds on several areas of prior work.

4.2.1 Motion Synthesis and Blending

There have been several efforts to synthesize new motions by applying constraints or drawing on sets of motion capture recordings.

Gleicher et al. [25] propose a formulation for synthesizing a motion sequence respecting a set of spacetime constraints. They pose it as a global optimization over a high-dimensional motion space. The optimization is non-convex and grows in complexity as the number of constraints increases. The problem of combining takes could also be modeled as this kind of global optimization with the spatial constraints coming from users' annotations and temporal constraints coming from an

alignment method like Dynamic Time Warping [57]. Instead, I exploit the structure of our problem to propose a much simpler solution which leads to a one-dimensional optimization that can be solved in realtime. Further, unlike Gleicher et al. [25], my method is guaranteed to closely follow the original takes and the annotations.

Arikan et al. [8] use a database of motion sequences (captured using standard motion capture systems) to synthesize a new motion sequence. Each frame in the motion sequences is first tagged with actions like walk, run, jump etc. The user then specifies the desired action tags on a timeline and the system computes an optimal sequence of motion frames satisfying the tags. Their algorithm is based on representing the motion frames as a complete graph and then finding an optimal path in that graph using dynamic programming [44]. In my case, the motion sequences are semantically bound to a script and also vary in length of time. Hence a complete motion graph does not correctly represent the allowed valid transitions between motion frames for the story. Instead, we use temporal warping to align all the motion takes and then work with a graph that respects the temporal ordering of the frames. Further, the montage in this graph is non-Markovian in contrast to Arikan et al. due to an additional path constraint that we found necessary for editing the takes. Hence I propose a different solution based on performing an inference in a semi-Markov Conditional Random Field (CRF).

Kovar et al. [43] look at the problem of blending multiple, annotated, human motion capture clips. They propose techniques for temporally and spatially aligning the clips and then do a per-frame weighted average to blend them. I have found that taking weighted averages of existing takes gives undesirable results. For example, if a user happens to give high weight to two distinct motions that overlap in time, then the average at their overlap will not resemble either of the original motions. Instead, I focus on piecing together sections from the original takes, respecting user annotations to the extent possible, while finding good places to transition between the takes.

4.2.2 Interactive Compositing of Photos and Videos

This work on compositing multiple motion takes is inspired by Agarwala et al.'s [3] *Interactive Digital Photomontage* system for combining the best parts of a set of photographs. The user roughly annotates the parts of each photograph that are desired in the composite and the system uses a combi-

nation of graph-cut optimization [15] with gradient domain blending [59] to automatically generate the composite image. Ruegg et al.'s [66] *DuctTake* system extends the approach of Agarwala et al. [3] to compositing multiple videos. The MotionMontage system similarly allows users to annotate the desired parts of each motion take, and automatically combines them into a motion montage. Since we are working with 3D motion data rather than images or video, I use very different methods for optimizing the final composite.

The remaining chapter is organized as follows. In Section 4.3 I describe how the playspace for MotionMontage is setup and the processing pipeline. I then describe how the interaction model through which the user creates montages for one or more objects in Section 4.4. Next, I mathematically formulate the problem of merging multiple takes and propose a novel algorithm in Section 4.5. I then report on a user study in Section 4.8 to evaluate the usefulness of montages. Finally I conclude in Section 4.10 by summarizing the contributions of this work and discussing future work in this domain.

4.3 Playspace Setup and Processing Pipeline

In this section, I describe how the playspace framework is used for the MotionMontage system. Figure 4.2 shows a user using the system. The user works on a planar surface which is under observation of a color+depth Kinect® sensor. The surface is demarcated into two regions – *Play Area*, where the user moves a physical object called a *proxy*, and one *Control Box – Like Box* through which user annotates the takes. The display screen renders a virtual object in the same pose as that of the tracked proxy. The keyboard and mouse are used to provide different input commands to the system. The user can set up this playspace easily as described in Chapter 2.

Any physical object can be be used as a proxy to control the virtual object, as long as a virtual model is available for tracking it. In this system, the user constructs the proxy with Duplo® blocks using the DuploTrack system 3, which then provides the model of the proxy to be used in tracking. Currently I assume that there is only one proxy object. However the playspace framework can be easily extended in future to load multiple proxy objects and automatically recognize the proxy at runtime using computer vision features.

MotionMontage fits seamlessly the software framework of playspaces (Figure 2.1) as an ap-

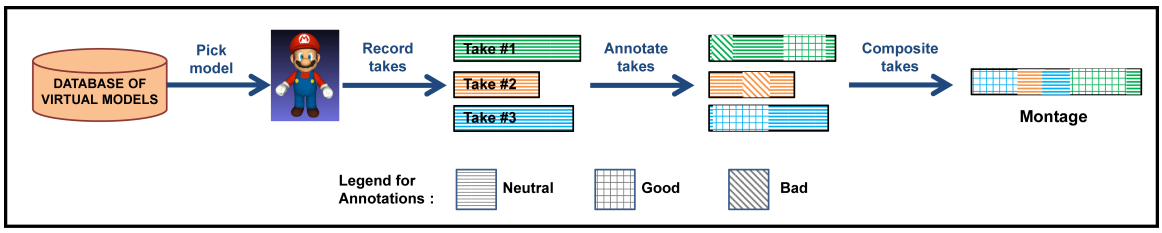


Figure 4.3: System functionality. The user chooses a virtual object from a pre-loaded database and records takes for the object. After that, he annotates the takes and the system combines them into a montage, which can be longer than any of the original takes. The process can be repeated for subsequent objects.

plication. The playspace algorithms from Chapter 2 provide a tracked pose for the proxy and the observed point cloud for the content in the *Like Box*. The system assumes that the *Like Box* is used as a slider using a green block. The median position of any green pixels in the point cloud is computed and scaled between 0 and 1 using the known width of the *Like Box*. This width is noted when the user defines the *Like Box* by clicking the four points corresponding to it while setting up the playspace (refer to Section 2.2.3). The controls from keyboard and mouse and mapped to appropriate controls through which the user records, navigates and edits the takes. I describe this interaction model in the next section.

4.4 System Functionality and User Interaction

The MotionMontage system allows the user to record 3D animations involving multiple objects, one at a time. I first discuss animating a single object, and then discuss differences in this procedure for animating multiple objects.

Figure 4.3 gives an overview of the system. The user first chooses a virtual object from a pre-loaded database. He then records one or more takes for the object, based on a script, by moving a physical object, called a *proxy*, in the 3D space in front of him. The tracked motion of the proxy is mapped in realtime onto a virtual object on a screen. The next step is to annotate each take, indicating which parts the user considers to be better or worse. The figure shows three discrete levels of annotations although there is a continuous scale. After annotating the takes, the system combines the best parts of each one into a montage animation.

The user can then repeat the process to animate additional virtual objects, and the system combines the montages of all objects to create the final animation. In the remainder of this section, I describe the details the various interactions described above. The supplementary video shows the effects of all these interactions in detail.

4.4.1 Setting up the Scene and Virtual Objects

The user first chooses a virtual background scene. The background scenes are created by placing freely available 3D objects, downloaded from the internet. The *Play Area* on the table is mapped to an area in the background scene. This mapping can be modified using keyboard-based controls. Also, a trackball interface can be used to modify the viewpoint of the virtual scene at any point of time.

The set of virtual objects is created by downloading instances from online 3D databases. It can also be augmented with any models that the user may already have from scanning physical objects or from other sources. The user can use keys on the keyboard to scroll through this set and select the object that he wants to animate.

4.4.2 Recording a Take

The user starts recording a take by pressing a key on the keyboard. He then acts out the motion with the proxy and ends the recording by pressing a key again. The user can record multiple takes with this process. These takes may vary in length. However, the assumption is that all the takes follow the same template story, and vary only in motion style. Arrow keys and standard playback options allow the user to scroll through the takes and review them.

4.4.3 Annotating the Takes

The user annotates each take to indicate their varying satisfaction with each part of each take. Annotations are made in realtime as the take is played. While the take is playing, the user moves a green Duplo® block in the *Like Box* to indicate the degree of like or dislike for the part of the take that is currently playing. The *Like Box* acts as a slider, ranging from -5 to $+5$, with the block serving as the thumb of the slider. Moving the block to the right side of the *Like Box* indicates *like*, while mov-

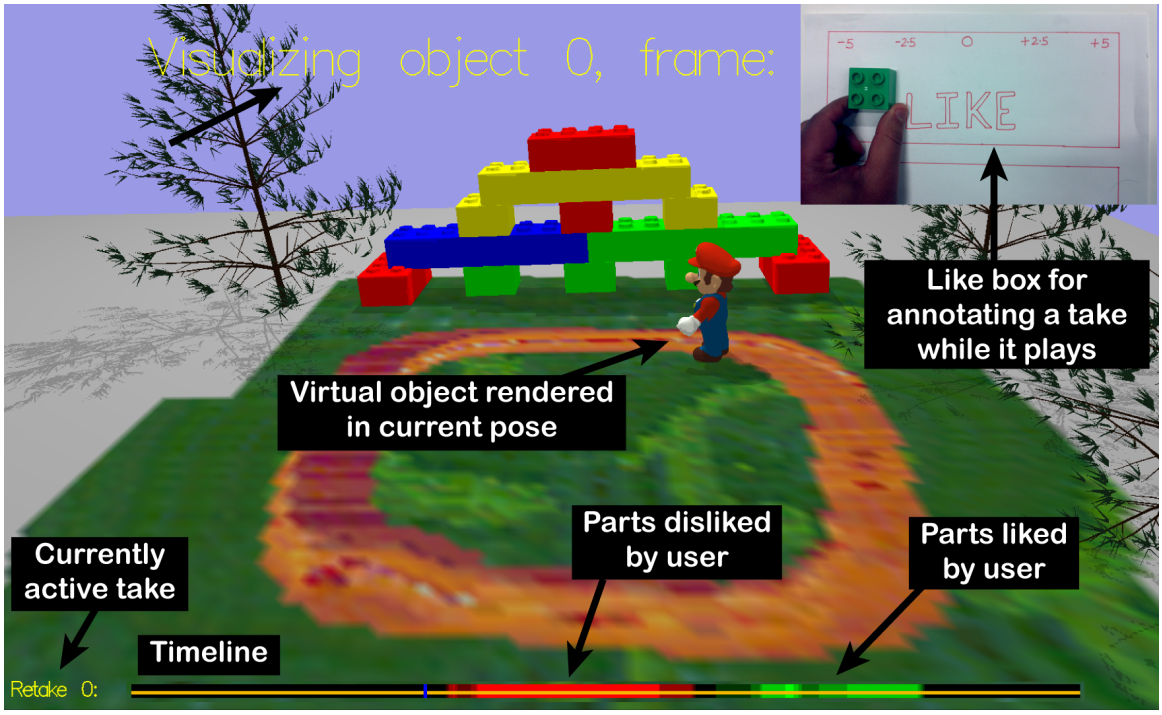


Figure 4.4: Annotating a take. While the take plays on the screen, the user moves a green block in the *Like Box* (top right). The system localizes the position of the block and annotates the take with the corresponding slider value. Disliked and liked parts of the timeline are illustrated with red and green, respectively. The degree of like or dislike, is reflected by the brightness of the color.

ing it to the left side indicates *dislike*. The tracked position of the block is re-scaled to the slider's scale. The user annotates the whole take in one play-through. Figure 4.4 shows a screenshot of the user annotating a take with this process. When annotating a take, the user sees a motion, judges it, and then physically moves the block. There is a natural time delay between the actual time of the motion and the moving of the block to its intended position. To compensate for this lag, we shift the recorded annotations back in time by 0.3 seconds. This value is based on the average human reaction time [1] plus an empirically added delay for block movement.

The user navigates through the takes using keyboard keys and repeats the annotation process for all the takes that he wants to composite for the montage. He then triggers the montage creation by pressing a key and the system generates the montage in realtime. I now describe a novel formulation and solution to this problem in the next section.

4.5 Compositing the Takes into the Montage

In this section, I discuss the formulation for merging m annotated motion takes into a single motion recording called a *montage*. I denote l^{th} take as T_l and its annotation as a function a_l where $T_l(t)$ and $a_l(t)$ give the pose and annotation values, respectively, at frame t. The annotation values lie on a continuous scale of -5 to 5. The pose $T_l(t)$ can be decomposed into a rotation quaternion and a translation vector, $T_l(t) = (q_l(t), x_l(t))$.

The goal is to create a montage that utilizes the best parts of each take without introducing noticeable artifacts due to switching between takes. The latter constraints mean we should find transition points where different takes agree, and we should avoid frequent flipping from one take to another. To determine the best set of switching points from one take to another, first all the takes are aligned in time. All the takes are temporally warped independently to match a single reference timeline. After warping, specific segments of each take are selected to create the montage on this timeline. Finally, the timeline is unwarped so that each segment will be played at its original speed, and then blend between adjacent segments over short intervals to create the final montage. We note that unwarping the timeline is important to respect the user's intentions; it is well known that timing significantly affects the meaning of an animated motion [79].

4.5.1 Temporal Warp of the Takes

The system uses a Dynamic Time Warping (DTW) [57] algorithm to align all the takes in time. For simplicity, we arbitrarily choose the first take as the reference, and warp all other takes to its timeline.

Formally, take T_1 is considered to be the *reference* take and then compute a monotonically increasing function w_l from the reference take's timeline to the timeline of each *source* take T_l. Each warp function w_l can be written as a mapping,

$$w_l : \{1, ..., ||T_1||\} \rightarrow \{1, ..., ||T_l||\} \tag{4.1}$$

which is monotonically increasing,

$$t_1 > t_2 \Rightarrow w_l(t_1) \geq w_l(t_2) \tag{4.2}$$

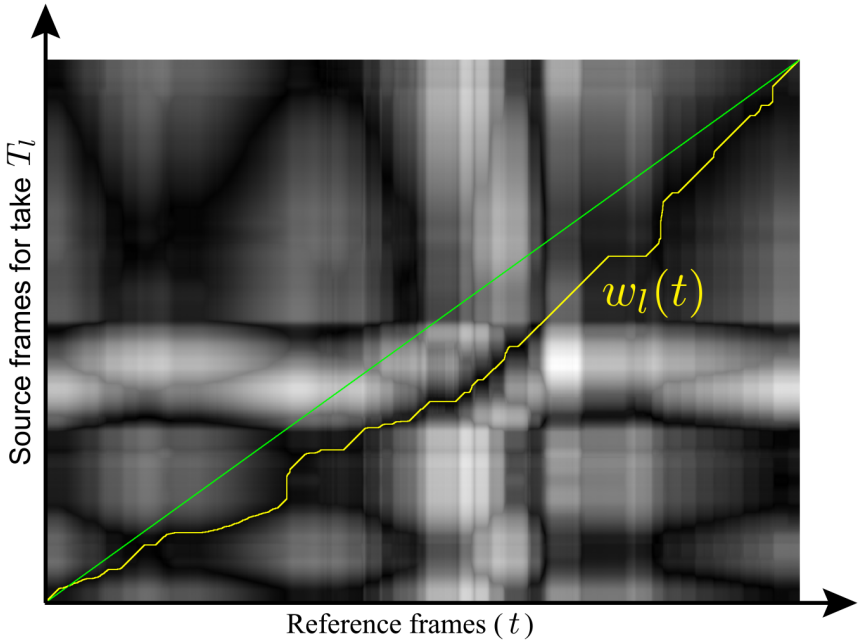


Figure 4.5: Dynamic Time Warping (DTW) finds an optimal warp function, $w_l(t)$, from the frames of the reference take, T_1 (x-axis), to the frames in the source take, T_l (y-axis). The values at each point in the quadrant represent the match of pose between the corresponding frames in the reference and the source takes. Darker color shows better pose match. The yellow path is optimal warp function computed by the algorithm while the green path shows the naive linear-scaling-based warp.

and matches the start and end frames of reference and source,

$$w_l(1) = 1 \quad \text{and} \quad w_l(||T_1||) = ||T_l|| \tag{4.3}$$

The system solve for an optimal w_l by minimizing the cost of bringing each reference frame into alignment with a source frame summed over all frames t:

$$w_l^* = \underset{w_l}{\text{argmin}} \sum_t C_{\text{match}} \left(T_1(t), T_l(w_l(t)) \right) \tag{4.4}$$

The matching cost C_{match} measures the difference between two poses. For two poses $A_1 = (q_1, x_1)$ and $A_2 = (q_2, x_2)$:

$$C_{\text{match}}(A_1, A_2) = ||q_1 - q_2||_2 + \lambda ||x_1 - x_2||_2 \tag{4.5}$$

λ is set to 2.5 based on experimentation.

Figure 4.5 shows a visualization of the matching cost and the optimal w_l. The standard DTW algorithm [57] is used to solve this problem. Graphically, it finds a monotonically increasing path that tries to stay in the darker regions of the diagram in Figure 4.5. $w_l(t)$ lets us warp $T_l(t)$ to a new warped take, $T'_l(t) = T_l(w_l(t))$ that follows the reference timeline.

We also need to warp the respective annotations to the reference timeline. This requires special care. Consider the case where $w_l(t)$ is constant for a duration of $N_{w_l(t)}$ frames. This corresponds to a horizontal, flat region of the curve in Figure 4.5. In this case, the annotation $a_l(w_l(t))$, which applies to only one frame in the source take, will have an influence over $N_{w_l(t)}$ frames in the warped space. As a result, during later optimization over the warped timeline (described in the next section), this annotation will have extra influence because it is artificially sustained by the warp. In this case, we should downweigh the annotation by $1/N_{w_l(t)}$ for each frame in the set that maps to $w_l(t)$.

Conversely, where the slope of $w_l(t)$ is steep, the warped sequence can skip from one source frame to a distant next source frame when moving from frame t to $t+1$ in the reference timeline, thus also skipping past all source annotations in between. The influence of these annotations will then be heavily undervalued; during later optimization over the warped timeline, these skipped frames will have no influence. In this case, we add all of their influences together to be stored as the warped annotation.

Formally, the warped annotations are computed as follows:

$$a'_l(t) = \begin{cases} \sum_{t'=w_l(t)}^{w_l(t+1)-1} a_l(t'), & \text{if } w_l(t+1) - w_l(t) > 0 \\ \\ \frac{1}{N_{w_l(t)}} a_l(w_l(t)), & \text{if } w_l(t+1) = w_l(t) \end{cases} \tag{4.6}$$

Another way of thinking about this is that we are (intelligently) scaling the values so that the summation of the warped annotations $a'_l(t)$ over $||T_1||$ frames of the reference timeline is the same as the summation of the unwarped annotations $a_l(t)$ over $||T_l||$ frames of the (unwarped) source timeline. I have found this scaling to significantly improve the quality of results over naively warping the annotation functions.

4.5.2 *Merging the Takes*

I now have m warped, annotated takes T'_l's, each comprised of $n = ||T_1||$ frames. I first seek to compute a warped montage, M', such that each frame of the montage comes from a corresponding frame in one of the m takes. M' can be denoted in shorthand as $\{l_1, l_2...l_n\}$ where $l_t \in \{1, 2...m\}$ is the take selected for time $t \in \{1, 2...n\}$. (Strictly speaking, the montage is a sequence of poses $\{T'_{l_1}(1), T'_{l_2}(2), ...T'_{l_n}(n)\}$.) I now define the overall cost function for a montage:

$$C_T(M') = \sum_{t=1}^{n} C_d(t, l_t) + \mu \sum_{t=1}^{n-1} C_s(t, l_t, l_{t+1}) \tag{4.7}$$

The total cost C_T is the sum of two terms: a data cost, C_d, and a smoothness cost C_s. μ is set to 100 based on experimentation.

C_d is the cost of frame t coming from take l_t and is based on the annotation functions, where a higher annotation rating translates to lower cost. Specifically, for a given frame t and take l I define this cost to be:

$$C_d(t, l) = -a'_l(t) \tag{4.8}$$

The smoothness cost C_s favors temporal coherence by discouraging transitions between takes in regions where their poses are dissimilar. At frame t, it is computed by considering time windows in the takes T_{l_t} and $T_{l_{t+1}}$, centered around the unwarped location of t, and summing up the cost of matching the poses in those windows. Specifically, given frame t and given two takes l and k, I define the cost to be:

$$C_s(t, l, k) = \sum_{h \in \mathcal{W}} C_{\text{match}}\left(\, T_l(w_l(t) + h),\, T_k(w_k(t+1) + h - 1)\,\right) \tag{4.9}$$

Note that poses of the unwarped takes are compared, as the final composite will be comprised of sequences of unwarped takes between which we want smooth transitions. The window $\mathcal{W}$ is set to the range $[-7, ..., 7]$, suitably truncated when the window goes out of bounds in either of the two takes. C_{match} is the same cost function as in Equation 4.5. Here we choose $\lambda = 0.8$ based on experimentation.

The optimal warped montage, M'^*, can be found by minimizing the total cost:

$$M'^* = \underset{M'}{\operatorname{argmin}}\, C_T(M') \tag{4.10}$$

This formulation is equivalent to an inference problem over a Markov chain, which can be solved exactly using the dynamic-programming-based Max-Sum algorithm [42].

However, I have found that optimizing this objective, despite the smoothness term, can occasionally lead to sections of the montage that have rapid, frequent flipping between annotated takes. To address this problem, I impose a minimum length on contiguous frames with the same label in the montage.

Specifically, the warped montage M' can be rewritten as a sequence of subsequences, $\{M'_1, M'_2, ...\}$, such that each subsequence of frames or *segment* $M'_j = \{l, l, ..., l\}$ comes from a single take l. I then impose the constraint that the length of each segment M'_j must be greater than a threshold d, set to 60 frames in our experiments. In further discussion, I refer to this additional constraint as the *Segment Length Constraint*. I also analyze the effect of this constraint on the montages created by the users in Section 4.8.3.

The formulation for the optimal montage M'^* now becomes,

$$M'^* = \underset{M'}{\operatorname{argmin}} C_T(M') \tag{4.11}$$

$$s.t. \qquad ||M'_j|| > d, \qquad \forall j \tag{4.12}$$

This formulation is equivalent to a Semi-Markov Conditional Random field that can be solved exactly with another dynamic-programming-based algorithm [69].

I now briefly describe the algorithm. Let $C(t, k)$ define the cost of the montage over the frames $\{1...t\}$ where $l_t = k$. $C(j, k)$ can be recursively defined as,

$$C(t,k) = \arg \min_{h \in \{1..(t-d+1)\}} \left(\operatorname{argmin}_{x \neq k} \Big(C(h-1,x) + C_s(h-1,x,k) \Big) + \sum_{g=\{h....t\}} C_d(g,k) \right) \tag{4.13}$$

This recursion finds the best interval, of at least length d from the end which can come from take k. It can be solved in $O(n^2m^2)$ time and $O(nm)$ space using dynamic programming where n is the number of frames and m is the number of takes. The optimal montage can be found by backtracking through the matrix C, starting from $C(n, l^*)$ where, $l^* = \operatorname{argmin}_l C(n, l)$.

4.5.3 Unwarping the Montage

M' is the optimal warped montage of length n frames where the frames come from the warped takes. This montage is now unwarped to preserve the original speed at which the takes were recorded. The span of frames for each segment M'_j in the warped montage is replaced with the corresponding interval of frames, M_j from the original take. Each corresponding interval is determined by the corresponding mapping function w_l. Thus, the unwarped montage, M, is a sequence of intervals of poses from the original takes T_l's.

4.5.4 Motion Blending

As a last step, the poses are blended around the transitions between takes in M to ensure temporal coherence. The system considers a window of length 15 frames centered at each transition point and blends the poses using a linear weighting scheme. For blending, it uses linear interpolation of the translation vectors and spherical linear interpolation of the rotation quaternions [74].

4.6 Creating montages for more than one object

I noted earlier that recording of takes for the first object are untimed, i.e., the user records the motion at his preferred speed. However, the takes of an added, new object cannot be untimed because its motion may need to be carefully timed with a previous object's recorded motion. Hence after the first object, I use fixed-time recording for the rest, i.e., all previous objects' motion play in realtime on screen as the new object's motion is recorded. This is similar to the layered animation recording approach of 3D Puppetry [30]. In the end, all the takes for new objects are of the same length. They can then be annotated and composited into the montage by the same process as described above, skipping the time warping stage.

Recording motion for a current object while the previously recorded motions of other objects play on the screen can be difficult since the user has to plan the current object's motion based on where other objects will be in future. For example, if the two objects were supposed to touch each other, they may end up passing by each other without touching or perhaps intersecting each other.

To aid the user, I introduce *spatial markers* to alleviate this problem. Spatial markers are static grey instances of the previously recorded objects rendered on the screen. The user adds a spatial

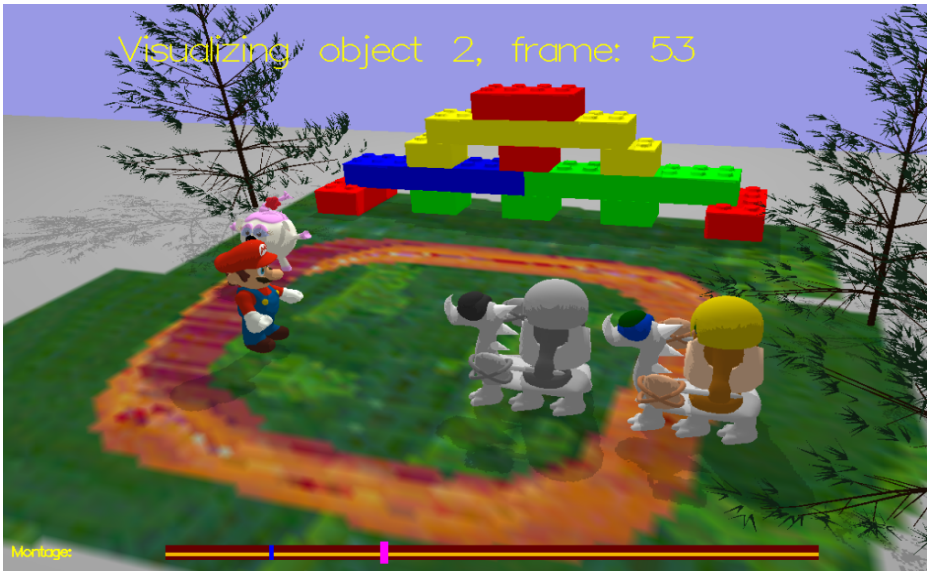


Figure 4.6: Mario (left, with red hat) is supposed to hit the Monster's head (right) in the script. The user adds a spatial marker for the Monster's position at the supposed time of the hit (3D replica in grey) to help plan Mario's motion. The marker's position in time is shown as a pink bar on the timeline.

marker for an object by navigating to a frame in the timeline of the object's montage and pressing a key. The spatial marker appears as a grey version of the object in that pose of the montage. It is also marked on the timeline as a pink bar. Figure 4.6 shows the usage of spatial markers in a sample script. Please see the supplementary videos to see the use of the markers in action. To avoid visual clutter, I do not allow two markers for the same object to be close to each other in time. This threshold is empirically chosen to be 60 frames. The user can remove the markers by clicking on them and using a key stroke. Spatial markers are essentially a sparsely sampled motion trail for an object where the sampling is decided by the user.

4.7 System Performance

The system runs in realtime on a desktop PC with two 6-core 3.33GHz Xeon processors and uses at most 400MB of RAM. To achieve this performance, the implementation is highly multithreaded with separate threads for processing the camera feed, tracking and rendering. The algorithm for merging the takes works in realtime. In my experiments, the length of takes can be on the order of

few thousand frames (few minutes at 30fps) and the number of takes on the order of 10. Creating a montage at this scale takes under a second. The computational complexity will increase with the increase in the size of the takes or the number of takes being composited. But I do not see that considerably affecting the performance of the montage-creation phase.

4.8 User Study: Single Object Montage

MotionMontage combines takes in a way that tries to preserve the parts most liked by the animator and discard the disliked parts, while keeping the whole animation temporally coherent. I conducted a user study to understand the performance of the system in three ways. First, I observed users creating and annotating the original takes. Second, I evaluated whether the montage is indeed perceived to be better than the original takes. Third, I analyze the quantitative and qualitative effect of *Segment Length Constraint* (from Section 4.5.2 on the montages.

4.8.1 Phase 1: Creating animations

The goal in the first phase was to familiarize the participants with using the system, then have them record takes and create a montage for a story script. I call the participants of this phase the *animators*.

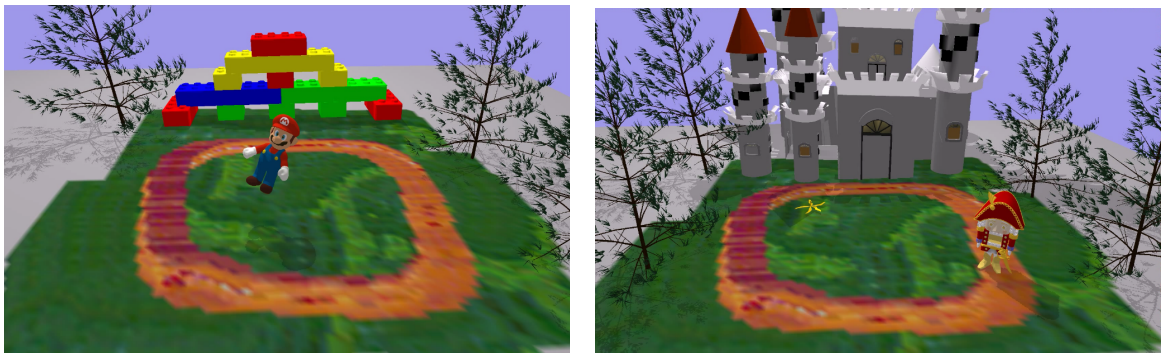
Introduction to System

The animators were first given an introduction to the hardware setup and the capabilities of the system. I then demonstrated the process of recording takes, annotating them, and creating a montage for a simple script –

Mario

"Mario is happily walking around the park. He first takes a round in a clockwise direction and then turns back and takes a round in an anti-clockwise direction."

Figure 4.7a shows a screenshot of this story. After the demo, the animators were asked to practice using the system by acting out this script, thus familiarizing themselves with the system.



(a) Demo script - *Mario*.

(b) Task script - *Soldier takes a Break*.

Figure 4.7: Screenshots of the scripts used for user study.

Task Design

After the demo, I asked the animators to record three takes for the following script –

Soldier takes a Break

"A soldier is guarding a castle by marching in front of it, back and forth. "This is so boring", he thinks. The weather is nice and he decides to take a walk around the castle's pathway. He looks around to see if anybody is watching him. Nobody else is there, so he starts his casual stroll. Slowly he becomes more and more carefree. Jumping around with joy, he does not notice a banana peel that is lying on his path. He steps on the banana peel, slips and falls down. He gets up slowly with effort and limps towards back to the castle. The monotonous routine starts again, guarding the castle by marching in front of it, back and forth, now with a limp. He finds it very hard to walk now and stops marching after reaching the other end."

Figure 4.7b shows a screenshot of this script. The animators were shown a sample path of the soldier and the goal was to record takes which roughly followed this path, while acting out the story script. I encouraged participants to try different motion styles on each take to better express the script. After recording the takes, the participants first viewed each take and then annotated them in a second viewing. The system used the annotated takes to create a montage.

To conclude, the animators filled out a short questionnaire about their experience with the system. I discuss this qualitative feedback later in the paper.

Participants

Twenty participants (ten female, ten male, ages 20 to 30) volunteered to create animations with our system. None of them had prior 3D animation experience. The study took about one hour for each participant. At the end of this phase, I had 3 takes and 1 montage for each of the 20 animators.

4.8.2 Phase 2: Comparing animations.

A week after the first phase I ran the second phase of the experiment. The goal in phase 2 was to compare the quality of the animations recorded in the first phase. I refer to the participants in this phase as *scorers*. Note that some of the scorers were also the animators in the first phase.

Task Design

I asked the scorers to perform binary comparisons between a montage and one of its takes recorded by a randomly chosen set of 10 animators. The scorers first read the script which the animators had acted out. In each comparison, the scorer watched one of the three takes and the montage in a random order. The scorer then selected the animation they would have preferred if they were the script's director. Note that scorers only rated one of the animator's takes against the montage instead of ranking all 3 takes and the montage. I designed the task this way to reduce visual fatigue for the scorer by simplifying the task to an A/B comparison. If a scorer was also an animator in Phase 1, I make sure that he did not do a binary comparison for his own animations.

If the scorer was also one of the animators, I added a second task. I asked them to rank all four animations (3 takes and 1 montage) that they created, ranking them from 1 to 4 (1 being the best) after watching them in a random order. This approach gave us a complete ranking of the animations from the animator's own perspective.

Participants

42 participants (21 female, 21 male, ages 20 to 30) volunteered for this phase. Of these, 20 were the animators from Phase 1. Each scorer's study lasted for about 50 minutes. At the end of the phase, I had 420 comparison samples, or 7 samples per comparison between a take and a montage for each animator. Additionally, I had each animator's ranking of their own takes and resulting montage.

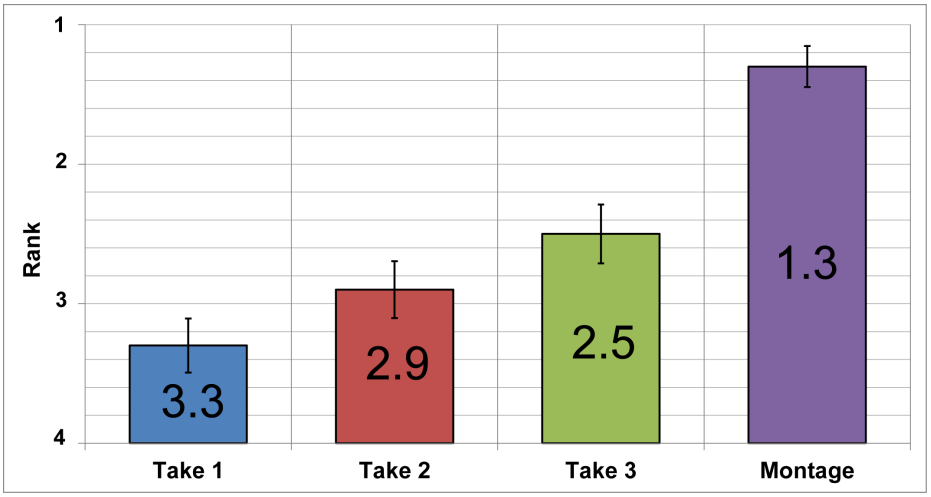


Figure 4.8: Average ranks for the takes (shown in the order originally recorded) and the resulting montage, as reported by animators evaluating their own work, averaged over the animators.

4.8.3 Results

I analyze the results of the experiment in four ways. First, I present a quantitative analysis of the perceived quality of montage from the animators' perspective. Second, I present a similar analysis about the perceived quality of montage from scorers' perspective. Third, I present some qualitative feedback from the animators about their experience with the system. Finally, I analyze if the *Segment Length Constraint* helps in creating better montages for the animators.

Animators' perspective

I analyze animators' rankings of their own animations (Take 1, Take 2, Take 3, Montage). I run a chi-square test with *rank* as an ordinal variable and *animation* as a nominal variable. *animation* is found to have a statistically significant impact on *rank*, $\chi^2(df = 3, N = 80) = 48.12, p < 0.0001$. Figure 4.8 shows the that the average rank of the montage, averaged over all the animators, is much better than any of the three takes. Thus, I can conclude that the animators significantly prefer their montages to any of their individual takes.

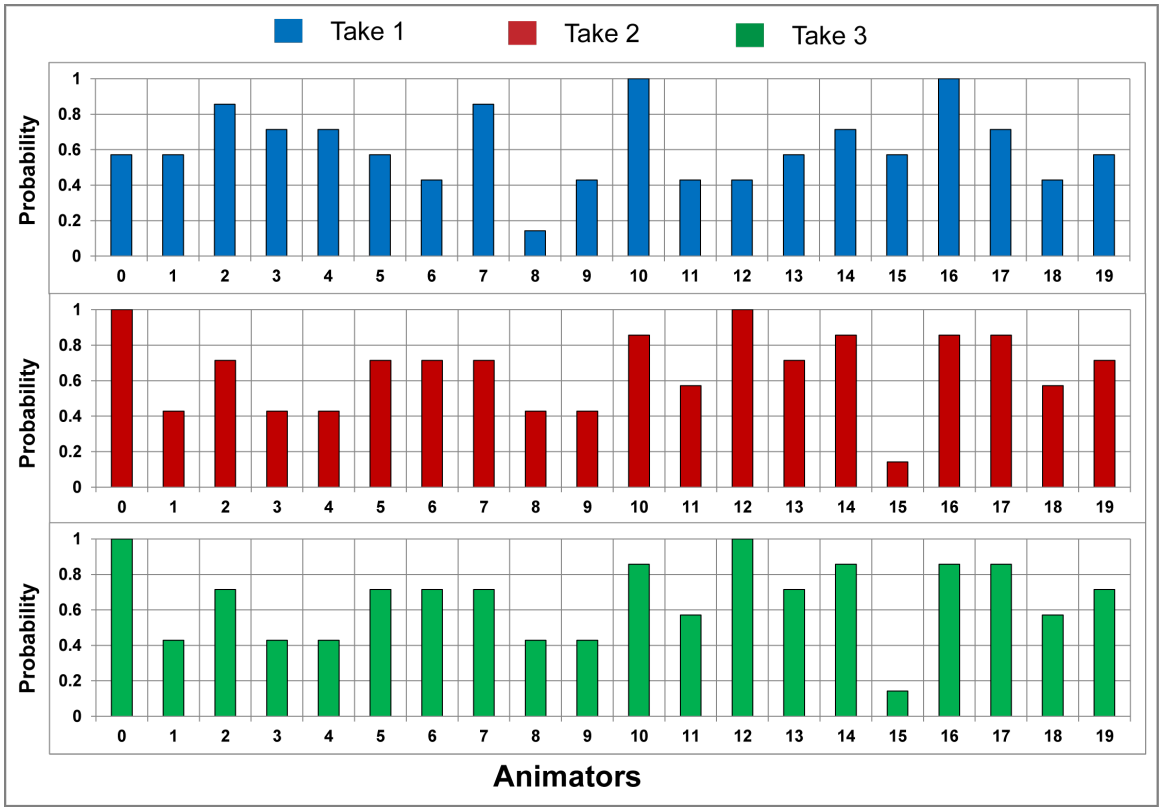


Figure 4.9: Probability that montage is better than a take for animators.

Scorers' perspective

Next, I analyze how scorers rate other people's animations, to see if they generally prefer montages to original takes. I first compute how frequently the montage is perceived to be better than an individual take. The proportion of the scorers who voted for the montage vs. each of the 60 takes is shown in Figure 4.9. For further analysis, I denote the fraction of the ratings that chose the montage for animator i better than take j as m_{ij}, i.e., if $m_{ij} = 1.0$ then all 7 scorers rated the montage better, and a score of 0 has the opposite meaning. I test three hypotheses.

H1. The average probability of an animator's montage being better than a take is greater than random chance, i.e., $\mu(m_{ij}) > 0.5$. A one-tailed single sample t-test indicates that the probability of the montage being better ($\mu = 0.6457$, $\sigma = 0.133$) is significantly greater than 0.5 ($t(19) = 5.18$, $p < 0.0001$).

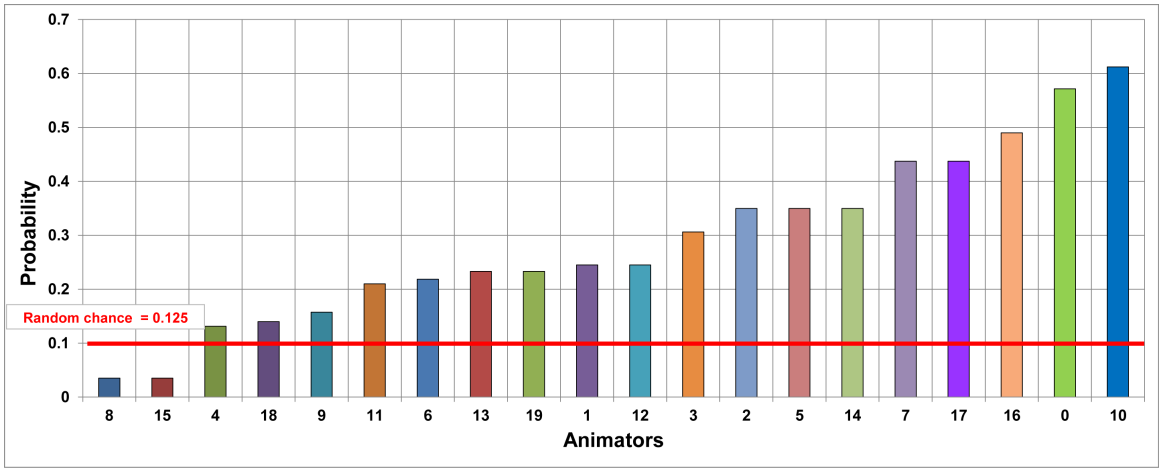


Figure 4.10: Probability that montage is better than all the takes for animators. The red line is the probability for random chance, 0.125.

H2. The probability of the montage being better than all the three takes for an animator is greater than random chance. This probability for an animator i is given by $(m_{i1} * m_{i2} * m_{i3})$, and I denote it as b_i. Thus the hypothesis can be restated as $\mu(b_i) > 0.125$. Figure 4.10 shows the probabilities b_i's for individual users. For all but 2 of the animators, the probability is higher than random chance. A one-tailed single sample t-test indicates that the probability of the montage being better than all the takes ($\mu = 0.299$, $\sigma = 0.126$) is significantly greater than 0.125 ($t(19) = 4.555$, $p < 0.0002$).

H3. The probability of the montage being rated better than an individual take significantly depends on the order of the takes, i.e. m_{ij} is related to j. Figure 4.11 shows the average probability and standard errors of the montage being better than individual takes averaged over animators. I conduct a mixed-model analysis with take j as the fixed effect, user i as random effect and probability m_{ij} as the observed variable. The analysis indicates that there is no significant effect of the order of take on the probability of montage being better, $(F(2, 38) = 0.8277, p = 0.4448)$. Hence we can reject the hypothesis.

Qualitative feedback

All the users said that they liked the capability to take multiple takes for the story, since they did not have any prior animation experience and wanted to try out different styles. They were excited

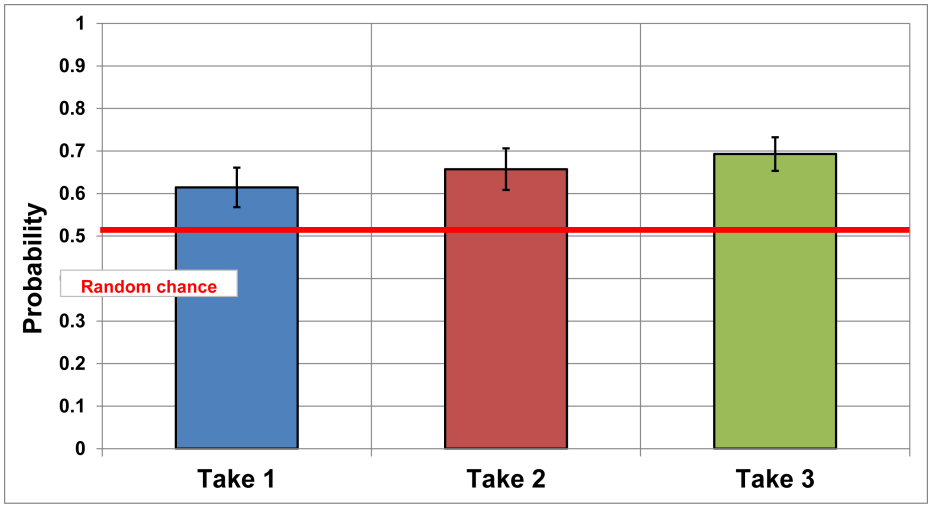


Figure 4.11: Probability that montage is better than a take (in the order they were recorded).

about the ability annotate different parts of the takes and that the system could combine the takes for them. A few users suggested that the method of annotations required patience as they had to watch the whole animations while annotating them. Further, they had to mentally remember the different parts from takes that they want to appear in the montage and annotate the takes accordingly. This leads to an interesting future direction of research of good ways to summarize and visualize the takes together.

Effect of the Segment Length Constraint *on resulting montage*

The Segment Length Constraint in computing the montage (Section 4.5.2) ensures that the minimum length of a contiguous set of frames taken from a take must exceed a minimum threshold. This threshold was empriically chosen to be 60 frames, i.e. a time duration of 2 seconds. The goal of this constraint is to enforce a restriction on quick take transitions in the montage.

For 10 of the 20 animators, not enforcing this constraint led to montage segments less than the threshold of 60 frames in length. Adding the constraint avoids such segments but results in a montage with a slightly higher cost. However, the increase in the cost was found to be less than 1% in all the 10 cases. Hence the proposed algorithm is able to enforce the Segment Length Constraint without sacrificing much on the cost of the montage.

The supplementary video shows a visual example of a discontinuity caused because of a short segment when the Segment Length Constraint was not enforced. We believe that the correct way to analyze the effect of this constraint will be to conduct a user study with A-B comparisons of montages with and without the constraint. We leave this study for future research work.

Results summary and discussion

The results suggest that the montages are significantly preferred both by the animators and the scorers. It is interesting to note that the average rank assigned by the animator is affected by the order in which takes were recorded. The rank is better for a take that was recorded later. I speculate that the animators, who were creating animations for the first time, grew better at acting out stylized motions with subsequent takes. However, the non-animator scorers' preference for the montage over the takes does not depend on their order to a statistically significant degree. This result does not support the conclusion that overall animation quality increases as animators gain more experience. I do not have a hypothesis for this difference and leave the understanding of visual or psychological perception in such animations to future work.

4.9 User Study: Multiple Objects and Markers

I conducted an informal user study to observe how users work with the creation of layered animations using our system. The layered animation allows the user to record motions for one object at a time. He can also put spatial markers for previously recorded object motions to help him plan the motion of current object.

Two users (one female and one male) volunteered for this study. They had been animators in the first phase of the previous user study. I first demonstrated to them the process of recording layered animations using our system. The users first practiced recording multiple takes with multiple objects.

For the actual task, I showed them a short animation involving three objects previously recorded by us the MotionMontage system. (Please see the supplementary video for the complete animation.) The task for the users was to record the object motions to roughly match the example animation. They were not allowed to use the montage capability here to simplify the task. I asked the users

to record one take for the first object and then two takes each for the second and third objects. They were not allowed to use markers in the first takes for the second and third objects, but were allowed to use markers for the second takes of these objects. After recording the takes, I asked them about the ease of sequentially recording takes for individual objects and whether the spatial markers helped them in planning the object motion.

I observed that the users quickly became comfortable with recording multi-layered animations after a few practice runs. In the actual task, the users did have some problems planning the motions without the use of markers. This led to objects intersecting each other when they would have just avoided or touched. The users said that using the markers helped them to plan the motion better.

4.10 Conclusion and Future Work

I presented a system, MotionMontage, which allows users to record 3D animations involving multiple objects, one object at a time. The user can experiment with different motion styles and trajectories and record multiple takes for each object. The system allows the user to temporally annotate each take to indicate which parts of which takes are considered better or worse. The system then uses a novel formulation to combine these annotated takes into a montage. The same process can be performed sequentially for all the objects in the animation. The recording of takes for more than one object is handled via the traditional layered animation approach. We introduced the notion of spatial markers which helps the user to refer to the recorded motions of other objects while recording the current object's motion.

I also reported on a user study to qualitatively and quantitatively measure the efficacy of MotionMontage from the animators' and the scorers' perspective. The results indicate that the montage is significantly preferred over the original takes by both. In addition, I reported qualitative feedback from a few users creating multi-object 3D animations. The users found the system intuitive and indicated that the spatial markers helped them to record object interactions better in the animation.

The system can be extended in multiple directions. I would like to add the capability of recording audio voiceovers in the system. Currently audio can be added as an after-effect using a standard audio-video processing software. However, the users gave feedback that they were actually humming and speaking in their mind while acting out the story, and would have liked to record audio

while animating. Looking at merging multiple audio takes would also be an interesting direction to pursue.

To further improve the richness of the 3D animation, I am interested in enabling articulated 3D characters in the animation. Difficulties in mapping the articulation of physical objects to a character's motion will require new intuitive puppetry interfaces.

Finally, I observed from the user studies, that animation creation systems should empower the user while being intuitive, with easy-to-use interfaces. This is challenging since our target is novice users who do not have any prior animation experience but want to tell animated stories. The current interaction design of MotionMontage can be extended in many directions to achieve this goal. For instance, in addition to a physical controller and simple keyboard/mouse-based controls, it could be useful to present a multi-modal interface involving voice, gestures, multiple physical controller objects, etc. In the future I would like to explore this interaction space and try to find a good trade-off between user empowerment and intuitive, easy-to-use interfaces for creating 3D animations.

Chapter 5

VIRTUAL 3D SCENE DESIGN IN PLAYSPACES

"The way to get started is to quit talking and begin doing." – Walt Disney

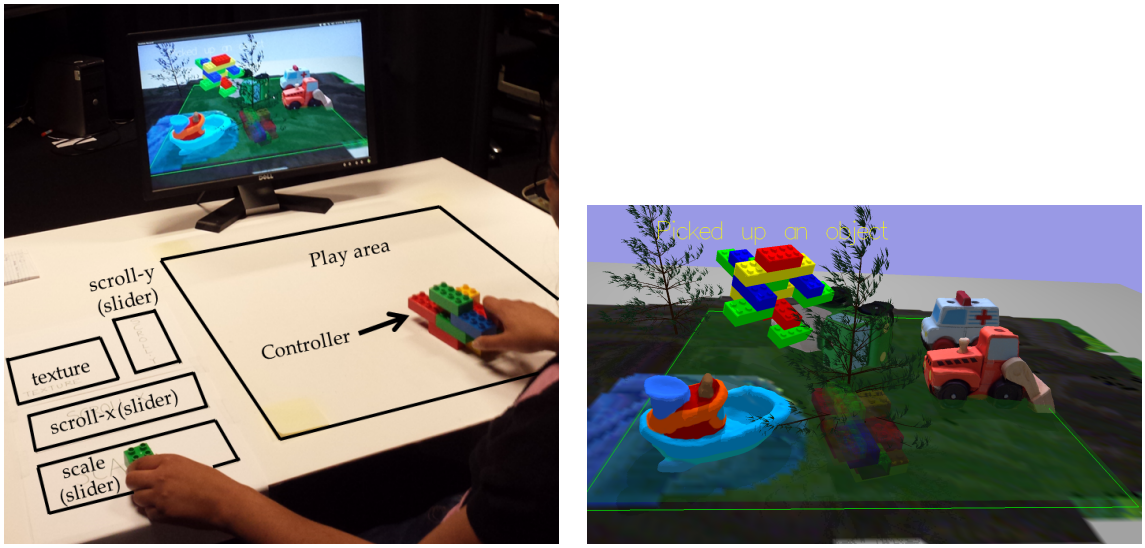
5.1 Introduction

Virtual 3D environments are used in a number of applications. Traditionally they are designed by expert digital artists for animations, games, interior home/office designs and city planning. These domains demand high quality and hence the corresponding CAD (Computer Aided Design) tools have complex interfaces with controls to design every fine detail. The complexity of these interfaces creates barriers for lay users wanting to try their hand at designing 3D virtual environments.

More recently, we have also seen games like Minecraft which trade off high fidelity design controls for the simplicity of the interface. Minecraft allows users to build 3D environments by placing and moving cubes in a 3D scene. The popularity of this game across age groups and the scale of 3D worlds created prove that users may not require high fidelity controls if the process is enjoyable.

The input interfaces for the aforementioned tools are still based on devices like keyboard, mouse and joystick. Further, the 3D environment under creation is rendered on a 2D display screen. Hence it is at times hard for a user to establish a clear mental mapping between the motion of the input devices to the positioning in the 3D virtual world. In this work, I develop a system, *SceneDesigner* through which lay users can design simple 3D scenes using an intuitive interface based on playspaces. The resulting 3D scenes can be used as is or for providing a conceptual seed to digital artists.

For the purpose of this work, I define a virtual 3D scene as a collection of object models placed on a planar terrain. Figure 5.1 shows a user using our system. The user works on a planar work-surface which is again divided into two regions – *Play Area* and *Control Boxes*. The *Play Area* is mapped directly to a part of the virtual terrain. The user works with a physical object as a *controller*



(a) A user using the SceneDesigner system.

(b) Closeup of the screen.

Figure 5.1: The user works on a planar work-surface which is divided into two regions – *Play Area* on the right and *Control Boxes* on the left. The *controller* object in the *Play Area* is tracked using a Kinect® camera and a translucent replica is rendered on the screen as shown in the screen close-up on the right. The replica can be attached to existing or new scene objects to manipulate the virtual scene rendered on the display screen. The user can use the *Control Boxes* to rescale objects, paint the terrain with a texture or navigate to different parts of the virtual scene. Here, the user has attached the controller to a virtual tree (seen better on the right) and is rescaling it using the green block in the *Scale Box* (seen on the left).

which is tracked in the *Play Area* through a Kinect® color+depth camera. A virtual replica of the controller is superimposed on the 3D scene that is rendered on the display screen in front of the user. The system allows the user to interact with the virtual scene using the controller and a combination of voice-based commands, keyboard/mouse-based commands, and through a set of *Control Boxes*. SceneDesigner currently provides two key functionalities –

- Re-arranging the scene objects: The user can attach the controller to existing or new virtual objects. A sequence of these operations can be used to add new objects, and move or delete an existing object in the scene. The attached virtual objects can also be rescaled using one of the *Control Boxes*.

- Painting the terrain: The user can use texture from a physical object to paint a part of the

terrain. The user can shows the texture to the camera which captures its image and attaches that to the base of the controller. The system then paints the terrain with the texture as the controller moves. Further, the height of the controller serves the purpose of increasing the brush size of this painting process.

The chapter is organized as follows. First I discuss some relevant prior work in Section 5.2. Next, I describe the system setup and functionality in Section 5.3. I then present some results and talk about system performance in Section 5.4. This work is currently at an early stage and can be extended into a lot of different directions. I provide a detailed discussion regarding these directions and conclude the work in Section 5.5.

5.2 Related Work

This work builds upon and uses techniques from a lot of domains. I now discuss the most relevant prior work in each of these domains.

5.2.1 Interfaces for 3D Modeling and Scene Design

Most of the existing commercial products like Sketchup® or Autodesk® 3ds Max® use keyboard/mouse based controls to design a 3D scene while it is rendered on the 3D screen. But it can be hard for lay users to start using these systems immediately because of the required need to master the input controls. Shin et al. [73] allow the users to sketch a view of the 3D scene and the system generates the scene by retrieving the objects closely matching the sketch and then allowing user to interactively specify the pose. However, sketching may not be a strong skill of lay users and further the system still needs the user to understand the 3D pose from a 2D rendering on the screen.

A key idea in this work is to provide a physical 3D space to the users that maps to the virtual environment and hence help develop a better perceptual understanding of the scene during the design process. There are 3D modeling tools that focus on using physical proxies [71] or augmented-reality style interfaces [18] for deformable and parametric objects respectively. However they do not look at the problem of arranging a 3D scene involving multiple objects. The closest to our work is the immersive authoring system of Lee et al. [47]. Their system uses an augmented reality-style interface to track 2D patterns and attach them to virtual objects to move them around. They do

not specifically look at the problem of designing a 3D scene but their interaction techniques of manipulating virtual objects is very similar to my work. They require the user to wear a headset and project the virtual scene directly on the physical pattens. This augmented reality-style setup may be perceptually better than my setup which is simpler to set up. There is scope to do a user study to compare these interfaces to understand users' preferences.

5.2.2 Methods for Painting a 3D Scene

Agarwala et al. [4] developed a system to paint a virtual replica of a physical object by moving a tracked sensor over the physical object. The paint corresponded to a fixed color picked from a digital palette. My system uses a similar approach for 2.5D terrains. It first captures the texture shown to the camera by the user, virtually attach it to the bottom surface of the tracked controller which then acts as a brush.

Ryokai et al. [68] developed a physical brush equipped with sensors which captures the texture of the object on which it is placed on using an embedded camera. The brush can then be used a physical brush to imprint that texture on a digital canvas. The pressure sensors govern the size of the brush. The texture painting idea for terrains is also very similar except that it is based in a different setting. The user shows the pattern to the static camera looking down at the table which then captures the texture and uses it for painting.

A major aspect of painting digitally is how to put the texture on the canvas, or terrain in our case. I take a simple blending-based approach where a copy of the texture is placed at each position of the brush and blended with the existing colors. This is a common approach taken while simulating layered painting. However this continuous blending approach leads to blurring of the texture in the direction of the brush. Ritter et al. [64] present a better approach where they use texture synthesis techniques to preserve the intricate structure of the texture while painting. In the future, I would like to integrate their technique for painting the terrain with textures. Applying this technique for non-planar terrains may give rise to interesting challenges.

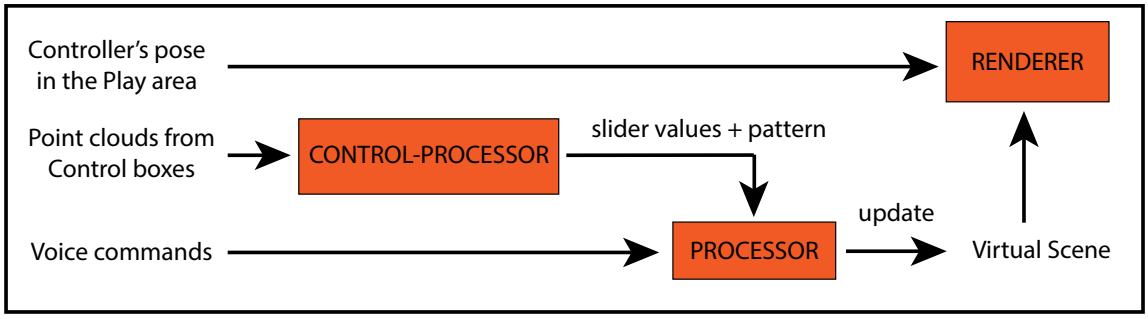


Figure 5.2: Processing pipeline for the SceneDesigner system.

5.3 System Overview

The SceneDesigner system works in a playspace framework as described in Section 5.1. The planar work-surface is divided into two areas – *Play Area* and *Control Boxes*. The application requires that the *Play Area* is roughly rectangular and that the relative layout of *Control Boxes* is as shown in Figure 5.1. This is important because of the way the system analyzes these boxes for inputs. Two of the edges of the *Play Area* need to be perpendicular to each other since they are used as the X and Y axes, with the work-surface's normal being the Z axis. The *Play Area* is directly mapped to a part of the virtual scene which is rendered on the display screen in front of the user. The user can move around a *controller* object in the *Play Area* and a tracked replica is rendered on the screen. The user uses voice-based commands – *pick*, *place*, *discard*, *copy*, *next*, *previous*, *texture*, *erase*, *back*, *write*, and four *Control Boxes* – *Scale*, *Scroll-x*, *Scroll-y* and *Texture*, to provide inputs to the system. The first three *Control Boxes* work as sliders and the *Texture Box* captures the pattern for painting the virtual terrain.

The SceneDesigner system seamlessly fits in as an application in the software framework of the playspace (Figure 2.1). Figure 5.2 shows how the outputs of the playspace algorithms from Chapter 2 are used by the SceneDesigner application. The 3D pose of the controller is used in the *Renderer* to render a virtual replica superimposed on the virtual scene. The point clouds from the *Control Boxes* are provided to the *Control-Processor* which extracts the slider values and the pattern. These are then passed onto the *Processor* module which decides the current mode of the system and the corresponding operations. The system works primarily in two modes – *Object manipulation* and

Terrain painting. In the Object manipulation mode, the user adds, moves, deletes or rescales virtual objects. In the Terrain painting mode, the user can show a physical texture to the system and then paint that texture on parts of the terrain. Besides this, the user can use the *Scroll-x* and *Scroll-y* boxes to translate the mapping of the *Play Area* in the virtual terrain to edit other parts of the scene which are not within the current boundaries. I call this operation *Scene Navigation*.

I now discuss each part of the system in more detail. First, I describe the system setup followed by the initialization and the internal representation of the 3D scene. I then describe how the Control-Processor module extracts the slider values and texture. Next, I explain how the Processor module transitions and functions in different modes thus allowing the user to manipulate the virtual scene.

5.3.1 System Setup

The user can easily set up this playspace using the algorithms described in Chapter 2. The virtual terrain is rendered in a canonical coordinate system and may not match the user's viewpoint. Hence as in the DuploTrack system in Chapter 3, the user needs to adjust the viewpoint on the screen using a virtual trackball interface so that the view of the *Play Area* matches the rendered view of the virtual scene.

Any physical object available to the user can be used as a controller as long as its virtual model is available to the tracking algorithm. The virtual model can be obtained by scanning the object or from an online repository.

In addition to this, the system loads a database of virtual object models. This database is built by downloading models from online repositories or by scanning physical objects that the user may have access to. There is a lot of scale and pose variation in the models downloaded from the internet. Hence all the models are re-scaled manually to be compatible with each other in size. They are also manually aligned to match the upright pose of the controller's virtual model.

5.3.2 Scene Representation and System Initialization

The virtual scene is represented as a planar terrain and a list of objects positioned in the 3D space above the terrain. The terrain's size is fixed to $5m \times 5m$ and its mesh model is sampled at the resolution of 1 point/mm. Each object stores a mesh model that lies at the origin, a scale factor

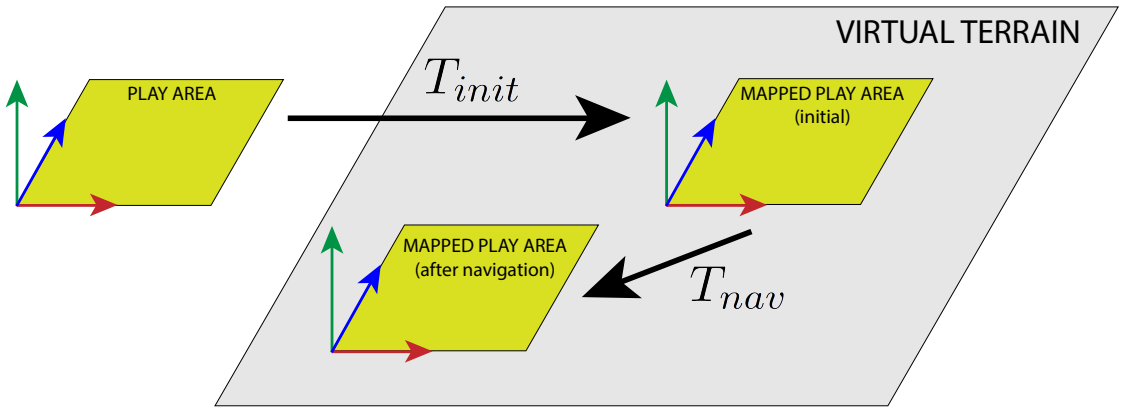


Figure 5.3: Mapping the *Play Area* to the virtual scene.

and a pose for positioning it in the scene. Thus a scene can be rendered on screen by rendering the terrain's mesh model and the objects' mesh models after applying the corresponding scale factor and the pose transformation.

The virtual scene is stationary and resides in a canonical coordinate system where the terrain lies on the X-Y plane and is centered at the origin. The positive Z axis forms the upward normal for the terrain. At start-up, the system computes a transformation T_{init} that maps the *Play Area* exactly to the virtual scene. The system assumes that one of the corners of the *Play Area* maps to the origin in the virtual scene and two edges emanating from that corner map to the X and Y axes such that the work-surface's normal maps to the Z axis. Figure 5.3 illustrates this mapping.

During the course of using the system, the user may want to move the *Play Area*'s mapping in the virtual scene to edit the other parts of the scene. He is allowed to translate the mapping along the virtual terrain in the X or Y direction. For this the system maintains a translation-only transformation T_{nav}. At start up, this is set to zero translation.

Hence at any point of time, the pose of the controller in the *Play Area* can be mapped to the pose in the virtual scene's coordinate system by applying the transformations T_{init} and T_{nav}, in this order.

At start up, the system is intialized with a virtual scene consisting of a planar terrain and zero objects. It can also be initialized with another scene previously created with SceneDesigner.

5.3.3 Control-Processor

The Control-Processor gets as inputs the point clouds in the *Control Boxes*.

The *Scroll-x*, *Scroll-y* and *Scale* boxes act as sliders along their longer dimension and are operated by a green block. Also, I restrict the usage of these boxes to one at a time. Amongst the three point clouds, the system selects the one of them which has sufficient and maximum number of green points. For the selected point cloud, the median position of the green points along the longer dimension of the corresponding box is computed. This value is then normalized w.r.t. the length of the longer dimension resulting in a value between 0 and 1. This is the corresponding slider's value. The threshold for sufficiency is chosen to be 20 points empirically. The classifier to check for green color is fairly simple – the green channel must exceed the red and the blue channels by a multiplicative factor of 1.5.

For the *Scale Box*, the slider value is further scaled to the interval $[0.2, 3.0]$, which is the range of rescaling allowed for an object. This mapping is a piecewise linear mapping which matches the interval end points and has a derivative discontinuity at 0.5 which is mapped to the scale of 1.

The *Texture Box* is used to capture the pattern that the user wants to paint on the terrain. The system captures a square patch centered about the center of the box and pass this bitmap image as the captured texture. The square window's dimension is either 50 or the lesser of the two dimensions of the pattern, whichever is smallest. These inputs are passed to the Control module which uses them in conjunction with the voice commands for further operations.

5.3.4 Processor

The Processor receives the voice commands and the inputs from *Control Boxes* and operates the system in different modes. The system also gives text feedback on the screen about the current mode. I now describe the functionality of each mode in detail. The supplementary video shows recorded demos of these functionalities.

Object Manipulation

To add a new object, the user says *pick*. The virtual replica becomes translucent and a virtual object model from the database is rendered over it in the tracked pose. The user can scroll back and forth

through the list of object models by saying *previous* or *next*. Once the desired object appears, the user can move the controller to the appropriate location in the *Play Area*. The user can optionally rescale the virtual model by using the green block in the *Scale Box*. The user can place the virtual object in the current location by saying *place*. The user can also decide not to place it anywhere by saying *discard*. The controller is back to being rendered in the opaque way. The new object is added to the list of scene objects with its pose in the canonical coordinate system and the scale associated with it.

To manipulate an existing scene object, the user moves the controller so that its virtual replica intersects that scene object on the screen. The user needs to be given some feedback when the replica intersects with a scene object. The system renders a bigger translucent version of the object over the actual object if the intersection happens. Once the controller's replica intersects with the desired scene object, the user says *pick*. The system attaches the scene object to the controller and follows its tracked pose. The user can move and optionally rescale the picked object and place it at a desired location by saying *place*. The user can also remove the object from the scene with the *discard* command. The system now goes back to the initial state.

The user can also create a copy of an existing object. He positions the controller in the *Play Area* so that its replica intersects with the desired object. The user says *copy* and the system attaches a copy of the object to the tracked controller. The user can move and rescale this new object and place it anywhere in the scene as described earlier.

Terrain Painting

To paint a part of the terrain, the user places a texture in the *Texture Box* and says *texture*. The system captures the patch from the *Texture Box*. The patch has square dimensions in pixels. To apply this patch it to the terrain, it needs to be converted to metric dimensions. The system maps the patch to a $4cm$ sized square area centered about the projected position of the controller's replica along the terrain normal. The size of the mapped area increases linearly to a maximum of $12cm$ with the controller's height.

To apply the patch to the area on the terrain, bilinear interpolation is used to resize the patch and then blend the colors at each point in the area using alpha blending. If C_{old} is the original color of a

terrain's point, and C_{patch} is the patch's color at that point, then the blended color is given by –

$$C_{new} = C_{old} * (1 - \alpha) + C_{patch} * \alpha \tag{5.1}$$

where α at each point is the value of a 2D Gaussian centered at the area's center and with a standard deviation of one-fourth the area's dimension. If the terrain was not painted earlier, i.e. C_{before} was not defined, then α is set to 1.0.

The user can come out of terrain painting mode by saying *back*. To erase texture from the terrain, the user says *erase* from the default state, and the same painting process as above takes place, except that now instead of putting color on the terrain area, the system removes the colors from those areas. The user comes out of this mode by saying *back*.

Scene Navigation

Since the virtual scene is bigger in area than the *Play Area*, the user is provided with a way to translate the *Play Area*'s mapping on the terrain to edit other parts of the terrain. The user can use the *Scroll-x* and *Scroll-y* boxes as sliders to move the mapping in X and Y directions respectively. The slider values lie in the interval $[0, 1]$ and are mapped to $[-5, +5]$ using a parabolic function where the minima lies at 0.5 which is mapped to 0. The new values refer to the velocity with which the mapping moves on the virtual terrain. The change in the mapping is reflected in the transformation T_{nav}, which stores the total amount of X and Y translation.

5.3.5 Saving the Scene

The user can save a 3D scene at any stage by saying *write*. The format of the saved scene is the list of objects with paths to the object's 3D models, and their scales and poses in the scene. It also links to the painted terrain model which is saved in standard PLY format as a separate file.

5.4 Performance and Applications

The system runs on a 12 core desktop machine in real-time. The segmentation, tracking, rendering and scene processing algorithms run as separate threads. The controller tracking has a bit of lag, as

in previous chapters, but it can be significantly sped up by implementing the ICP (Iterative Closest Point) 3D alignment algorithm on the GPUs as discussed in Chapter 2.

Figure 5.4 shows a few scenes that I created using the system. Some of these scenes were also used in the MotionMontage work from Chapter 4 to create digital animations. Besides creating sets for digital animations, this system can also be used for developing prototypes for interior home/office design or professional animations that the expert artists then work on. Further, the system can be extended to be used in collaborative environments for design or competitive game play.

The current version of the system is a just a first step towards exploring interactive environments for designing 3D virtual scenes. There is ample room for further research and development in a lot of different aspects of the system. I discuss these in the next section.

5.5 Conclusion and Future Work

I have developed and described an initial version of the SceneDesigner system which allows lay users to create virtual 3D scenes using a novel interface based on playspaces. A marked region on the planar work-surface of the playspace is mapped to the virtual scene. The user can use a physical object as a controller to navigate the mapped area and add, remove or move virtual objects in it. The system also allows the user to paint the planar terrain with a texture that can be shown to the camera. All along, the virtual scene with the controller's replica and appropriate visual feedback is rendered on the screen to assist the user.

As mentioned earlier, this is only a first step in designing a simple, yet powerful system to build 3D scenes. There are many interesting directions to work on in the future.

5.5.1 Not Just One Controller

My vision for this system is where the user can use multiple physical objects in the *Play Area* to represent different objects. These objects can be as small as chess pawns and as simple as a color piece of folded paper. So the user could just write *table* on a piece of paper, fold it and make it stand in the *Play Area*. The system could possibly detect the writing and retrieve a virtual table model in the scene or the user can choose from the database manually as is the case now. Tracking such

(a)

(b)

Figure 5.4: Example scenes designed using the SceneDesigner system.

objects in real-time will present interesting challenges.

Having multiple objects as controllers will result in a tangible scene arrangement in the *Play Area* in addition to the virtual world. This will provide a better spatial understanding about the scene compared to the current method of working with one controller and observing its effect on the display screen.

5.5.2 Retrieving Virtual Objects from Database

Currently, the user linearly scans through the objects of the virtual database by speaking *next* or *previous*. This can take a long time if the size of database is big. Hence there is a need to develop interfaces for quick retrieval of desired objects. One solution could be to tag the object models and retrieve the objects with the tags spoken by the user. Another solution could be to allow users to build customized controllers using Duplo® or Lego ® blocks and retrieve similar models as has been done by some researchers [85, 34]. A naive solution based on mouse-based pointing and clicking the desired virtual object from a collection of icons may also work well.

5.5.3 Viewing the Scene from Different Angles

To get a better understanding of the virtual 3D scene, it is important to be able to view it from different viewpoints smoothly. Currently, the user can use virtual trackball interface to move around the scene on the screen. However, this creates a disconnect in the ongoing design process. It might be a better idea to use a physical proxy object for the camera which is tracked in the 3D physical space and can help set the viewpoint on the screen. Another solution would be to remove the 2D display and let user be in an immersive environment where he can walk around in the scene and use augmented-reality style glasses to see the virtual world.

5.5.4 Geometric Manipulation of the Terrain

The current system only allows the user to paint the planar terrain with texture. The capability to geometrically edit the terrain by treating it as a 2.5D map would also be a significant enhancement to the system. A possible solution would be to add a terrain manipulation mode where some gesture of the controller could be mapped to increasing or decreasing the terrain height in a local neighborhood.

This will present interesting problems about handling the existing objects that stand on that part of the terrain or deforming the texture in that area in an appropriate way.

5.5.5 Collaborative SceneDesigner in Immersive Environments

In the future, an interesting direction would be to allow multiple people to design a system in an immersive environment. This will possibly involve tracking the human body and many other objects and design holographic technologies that project virtual objects in the physical space. Remote collaboration between people would also be an interesting direction to look into.

5.5.6 User Studies for Evaluation

A very important aspect of this work will be to evaluate it against the existing scene designing systems. The user study would look to test the hypothesis that the interfaces presented in this work allow lay users to create 3D scenes more easily than using a keyboard/mouse-based systems like Sketchup® or Lego Digital Designer®.

Chapter 6

CONCLUSION

"Now this is not the end. It is not even the beginning of the end.
But it is, perhaps, the end of the beginning." – Winston Churchill

6.1 Contributions

In this dissertation, I presented an easy-to-setup environment called *playspaces* which enables the users to interact with virtual 3D content in a more natural way. The key motivation behind this work is to allow the users to perform tasks in the physical world while still interacting with the virtual counterparts for assistance or content transfer in a seamless way. This way of interaction is better than the traditional interfaces for interacting with 3D virtual content which either use specialized augmented reality setups or require a steep learning curve on users' parts.

In a playspace, the user works on a planar work-surface while a color+depth camera looks down at it. Part of a planar work-surface called *Play Area* is mapped to the virtual world and any physical objects in the *Play Area* are tracked in real-time. The virtual world, which is rendered on a screen, can be manipulated by these physical objects as defined by the application running in the playspace. The framework also enables the user to define *Control Boxes* on the work-surface by a simple clicking mechanism. These boxes are monitored by the camera and can be used as buttons, sliders, or gesture-inputs using physical tokens or hands. Further, the framework integrates the usage of voice commands and inputs from standard devices like keyboard and mouse.

The modularity of the playspace's software framework allows applications to easily plug in and provide users with a richer interaction experience. I designed and developed three applications which deal with different types of 3D content – block model assemblies, digital storytelling and design of virtual scenes.

6.1.1 Playspaces for Block Model Assemblies

I presented a system *DuploTrack* in Chapter 3 that automatically learns and builds a virtual replica of a Duplo® block model by observing the user build it. It also assists the user in creating a predefined model in a novel way while detecting any mistakes and assisting in making any corrections on the fly. I reported on a user study that shows that the proposed guidance method is better than the traditional figure-based guidance method.

6.1.2 Playspaces for Digital Storytelling

I presented a system *MotionMontage* in Chapter 4 that allows a user to act out a story using rigid puppets and automatically creates an animation from the tracked puppet motion. Further, it allows the user to record multiple takes for the same story and merge them automatically into a montage after the user has roughly annotated them based on his liking. This approach is helpful when the user wants to try out different styles and later merge them. I reported on a user study that shows that lay users are able to work easily with the system and that the created montages are perceived to be visually better by the creators as well as viewers.

6.1.3 Playspaces for Designing Virtual 3D Scenes

I presented a system *SceneDesigner* in Chapter 5 that allows the user to easily design simple 3D virtual scenes. Instead of using a traditional keyboard/mouse-based interfaces the user manipulates virtual objects in a scene by attaching and detaching them to/from physical objects of his choice. The user can add, move, scale, clone or delete objects from a database, thus creating simple 3D virtual environments. The system also allows the user to paint the terrain in the virtual world by using textures from his surroundings.

In all the above applications, I found that the playspace environment was more engaging than traditional interfaces. I identified current limitations of each of the systems and proposed some ideas to overcome them. I now talk about some future research directions for advancing the concept of playspaces and for each of the above applications.

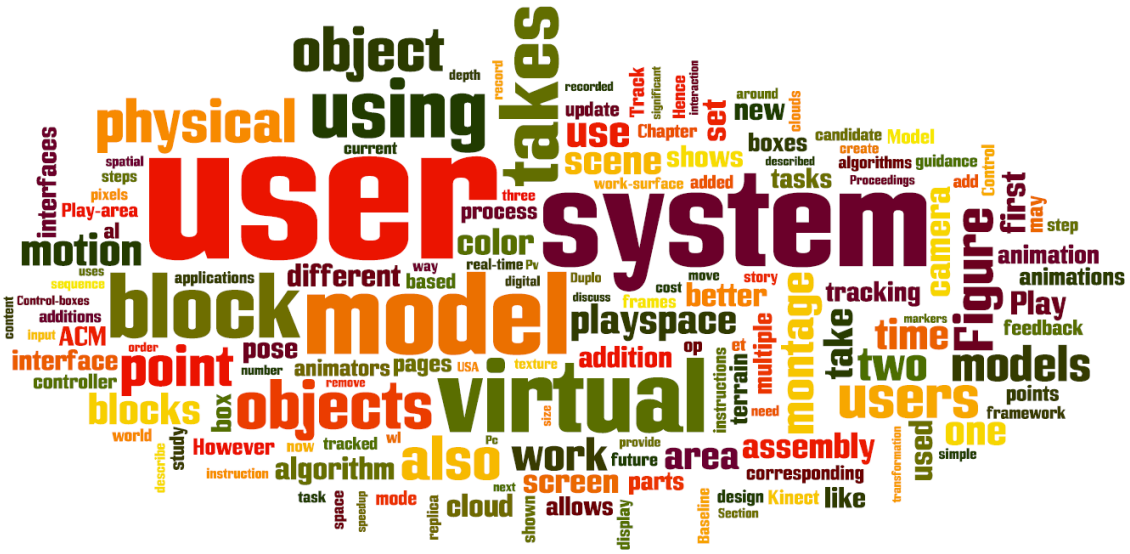


Figure 6.1: Word cloud summary of the dissertation.

6.2 Future Work

The current design of playspaces is only one exploratory step in the domain of interactive environments. I have identified some of its limitations by observing the users using playspaces. First, having the virtual content on the 2D display screen still requires the user to look back and forth between the work-surface and the screen. Augmented-reality (AR) style techniques bridge this gap but traditionally those technologies have been quite expensive and specialized. However, we see are beginning to see some AR systems commercially these days – Google Glass, zSpace to name two. In such frameworks, there will be no need to have a static color+depth camera or a display screen. Both will probably sit around the human eye. Such systems will present their own challenges in terms of tracking objects successfully and projecting virtual content wthout lag.

A playspace allows the use of multiple modalities – visual, voice, gestures and standard input devices. The application running in the playspace specifies the exact usage of these modalities. However, in the future I want to explore systems where the application just defines operations and the user is able to define his personalized interface by combining all these different modalities. This is motivated by the idea that all human beings are unique and have different comfort levels and skill

sets in using different modalities. Some people might be dexterous with hands while others may be good at articulating things with their voice. Hence a user should be allowed to easily define his own interface for the task at hand.

I now discuss the future work in the two main domains that this dissertation explored – object assemblies and digital storytelling.

6.2.1 Object Assemblies

In this dissertation, I limited myself to working with assemblies of Duplo® block models. This choice was mainly because even with this simpler case, the problem was challenging in terms of developing robust ways to track objects and infer any structural changes.

In the future, I want to look at assisting users with objects assemblies which are much more complex with large variation in part shapes and sizes, and even including articulated parts. Assembling furniture, Legostorm® robot kits and desktop computers are a few common examples of such tasks. Building an interactive system to track and assist people through these tasks is a lot more challenging. First, it may not possible to do this with a static camera as was the case in DuploTrack. We would probably need an egocentric camera or a mobile camera which can get multiple views of the task easily. Second, the tracking and inference mechanisms will need to be much more sophisticated given the large variety in part sizes and shapes. Third, a real-time assistance framework may be too expensive. Instead a query based model for task assistance might work well. In this query-based model the camera would not track each activity in real-time but only try to check for correctness and then assist when the user asks for it.

Overall I believe the domain of object assemblies is very exciting and has many technical challenges in store that span a variety of fields including computer vision, computer graphics, machine learning and human computer interaction.

6.2.2 Digital Storytelling

The use of playspaces for digital storytelling has opened a completely new domain for users to convey their stories using any physical objects around them. These tools could also be used by professional directors to convey their ideas to digital artists in a more tangible way and hence improve

the workflow. Currently these tools are still limited to tracking objects that are rigid and are sufficiently but not too big. With the advances in sensing technology and computer vision algorithms in future, I foresee these limitations going away. I would like to point out a few key ideas and directions in this domain that can benefit from future research and can have a big impact –

- Tangible cinematography. The work in this dissertation only looks at tangible ways to record the animation. However, camera view angles and transitions play a key role in presenting any story. James Cameron used a virtual camera for shooting the movie *Avatar* which he could physically carry around, visualizing the rendered world, and hence plan the camera path. Developing similar accessible technology for lay users for editing camera views in their own recorded animations is an interesting future work direction.

- Gestural interfaces for style editing. In my MotionMontage system, the user needs to re-enact the new style that he wants to try out and then later combine takes using annotations. However, it may be much easier to provide high level style inputs by waving one's hands or even through facial expressions. Finding a mapping from this abstract gesture space to the 3D motions of objects is a challenging task and involves understanding user's intent. However, it has huge potential in making digital animation editing more accessible to the lay users.

Digital stories are a great way to convey ideas visually, and tangible interfaces such as the ones presented in this thesis make this art accessible to lay users. The goal of future research in this direction is to provide the means to create and edit higher quality animations while making the underlying technology invisible.

6.3 Summary

In this dissertation, I designed, developed and evaluated the playspace environment specifically for interacting with 3D virtual content. The design of the playspace was motivated by the ease of setting it up and flexibility in its usage. I demonstrated the efficacy of this concept through three applications targeted towards working with 3D content.

I strongly believe that technology should not be a replacement but a means of empowerment for human capability. This empowerment can come in forms of assistance in daily tasks or seeing

creative ideas come to life. For instance, the lay users who used my MotionMontage system had never made an animation before but soon they were adding their own motion effects and twists to the storyline. They exclaimed in joy now and again seeing their creative ideas come to reality seamlessly. It is this joy of using technology that I want to target with an idea like playspaces. However, playspaces are just one small step towards exploring this domain. I hope this dissertation motivates researchers and opens doors for exploring novel interaction environments for 3D content that are joyful and useful at the same time.

Appendix A

REORDERING ASSEMBLY STEPS TO ACCOUNT FOR BLOCK REMOVALS

A.1 Problem Definition

We are given a sequence of block addition or removal operations for building a block model. It is known that the model remains one connected piece at any stage during this building sequence. We need to compute a new sequence of operations to build the same model such that no block removals occur. Basically, we do not want to add the blocks which are going to be removed later.

The general version of the problem is to generate assembly instructions given the structure of a block model. This is useful for toy companies like Lego® who sell assembly sets for building object models. Allerelli et al. [36] have patented a technique which computes an optimal assembly sequence by dividing the model into sub-assemblies and computing an optimal sequence of assembly steps for each.

In my problem, I am interested in keeping the building process as close to the original sequence of operations through which it was built. Hence I propose an algorithm which first removes all the redundant addition operations. An addition operation is redundant if the corresponding block is removed from the assembly at a later stage.

However, the new sequence may have block additions which are not possible at that stage. Figure A.1 shows an example of this. In this example, the blocks are first added in the order - 1, 2,

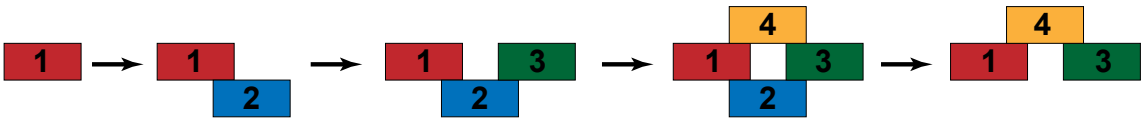


Figure A.1: Example where assembly steps need to be reordered. The sequence of steps are – Add 1, Add 2, Add 3, Add 4, Remove 2. After removing the redundant addition of 2, the steps become – Add 1, Add 3, Add 4. However, 3 cannot be added in this order since it has no connection to 1. The correct order would be to postpone addition of 3 after addition of 4.

3 and 4. Then block 2 is removed. If we remove the redundant additions, the remaining sequence is 1, 3, 4. Clearly 3 cannot be added be added right after 1 because there is no connection. Hence the addition of block 3 has to be delayed till after block 4. This leads us to the general strategy or reordering where we postpone the addition of disconnected blocks till we find a connection for them in the subsequent steps.

I now describe the algorithm formally.

A.2 Algorithm

Let S be a sequence of block additions, implemented as a queue, after removing the redundant additions. Each block addition is uniquely identified by the voxel it occupies. Let the new sequence of additions be denoted by R, again implemented as a queue. R is initialized with the first element of S which is popped from S. We linearly scan elements from S, process them and continue till S has been exhausted.

A.2.1 Invariant state

The algorithm maintains the following invariant state. Consider a stage where we have read the first i block additions from S. R has j out of those block additions which can be connected together. We maintain the unconnected block additions so far, $i - j$ in number, in a graph G where the graph nodes are the block additions and the edges indicate connections between those blocks. The nodes of G also have an ordering associated with them based on when they were added to G.

A.2.2 Each iteration

At any stage, the algorithm reads a block addition b from S.

If b can be connected to the elements in R, then it is pushed in R. We then check if any nodes of G have a connection with b. We expand this set of nodes to all the nodes of G which are connected to these nodes. This computation is done via breadth first search in G starting with these nodes. The nodes from this expanded set are deleted from G and pushed into R in the order in which they had been inserted into G.

Else, when b cannot be connected to elements in R, it is added to G. The connections of b with other nodes of G are added as new edges.

A.2.3 Correctness

Let us assume that the algorithm is incorrect. There can be two cases.

Case 1: There exists a block addition in R which is not connected to its preceding blocks. This is impossible by the algorithm's definition because a block addition is only added to R if it is connected to the preceding blocks.

Case 2: The algorithm ends with G non-empty. This implies that the algorithm has divided the block additions into two disjoint sets R and G of blocks with no edges between them. However we know that the physical model is connected at the end and hence the blocks cannot be divided into two disjoint sets. Hence either R is empty or G should be empty. We started with one block in R so G has to be empty. This contradicts the assumption that G is non-empty.

Thus the algorithm correctly results in a sequence of block additions such that the any subsequence of steps starting from the first block addition is a connected model.

A.2.4 Complexity

Processing each node can involve two cases – addition to R or addition to G. Checking the connection of a block b with elements in R can be done O(1) time by maintaining a voxel occupancy grid and just checking if the voxels above or below b are occupied for a valid connection. Hence the cost of adding b to R is the cost of doing breadth first search over the nodes connecting with b and adding them to R. Since all those nodes are deleted from G, the total cost of breadth first searches over the course of entire algorithm in $O(n)$ where n is the number of block additions. Similarly, the cost of adding b to G is also O(1) since we can check for any connections of b with nodes of G by maintaining a voxel occupancy grid for G.

Hence the total computational complexity of the algorithm is $O(n)$, where n is the number of block additions.

BIBLIOGRAPHY

[1] Human benchmark - reaction time statistics. www.humanbenchmark.com/tests/reactiontime.

[2] zspace: A natural 3d experience. "http://zspace.com".

[3] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 294–302, New York, NY, USA, 2004. ACM.

[4] Maneesh Agrawala, Andrew C. Beers, and Marc Levoy. 3d painting on scanned surfaces. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pages 145–ff., New York, NY, USA, 1995. ACM.

[5] Maneesh Agrawala, Wilmot Li, and Floraine Berthouzoz. Design principles for visual communication. *Communications of the ACM*, pages 60–69, April 2011.

[6] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. In *Proc. SIGGRAPH '03*, pages 828–837. ACM Press, 2003.

[7] David Anderson, James L. Frankel, Joe Marks, Aseem Agarwala, Paul Beardsley, Jessica Hodgins, Darren Leigh, Kathy Ryall, Eddie Sullivan, and Jonathan S. Yedidia. Tangible interaction + graphical interpretation: a new approach to 3d modeling. In *Proc. SIGGRAPH '00*, pages 393–402. ACM Press/Addison-Wesley Publishing Co., 2000.

[8] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3):402–408, July 2003.

[9] Autodesk. 123d catch. http://www.123dapp.com/catch/.

[10] Connelly Barnes, David E. Jacobs, Jason Sanders, Dan B Goldman, Szymon Rusinkiewicz, Adam Finkelstein, and Maneesh Agrawala. Video puppetry: a performative interface for cutout animation. *ACM Trans. Graph.*, 27(5):124:1–124:9, December 2008.

[11] Hrvoje Benko, Ricardo Jota, and Andrew Wilson. Miragetable: freehand interaction on a projected augmented reality tabletop. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 199–208, New York, NY, USA, 2012. ACM.

[12] P.J. Besl and N.D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:239–256, 1992.

[13] Richard A. Bolt. p̈ut-that-there̋: Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA, 1980. ACM.

[14] A. C. Boud, David J. Haniff, Chris Baber, and S. J. Steiner. Virtual reality and augmented reality as a training tool for assembly tasks. In *Proc. IV '99*, pages 32–36. IEEE Computer Society, 1999.

[15] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.

[16] M. Beth Casey, Ellen Winner, Mary M. Brabeck, Kate Sullivan, Kenneth J. Gilhooly, Mark T. G. Keane, Robert H. Logie, and George Erdos. Visual-spatial abilities in art, maths and science majors: Effects of sex, family handedness and spatial experience. *Lines of thinking: Reflections on the psychology of thought*, 2:275–294, 1990.

[17] Jiawen Chen, Shahram Izadi, and Andrew Fitzgibbon. KinÊtre: animating the world with the human body. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 435–444, New York, NY, USA, 2012. ACM.

[18] Bruno R. De Araùjo, Géry Casiez, and Joaquim A. Jorge. Mockup builder: direct 3d modeling on and above the surface in a continuous interaction space. In *Proceedings of Graphics Interface 2012*, GI '12, pages 173–180, Toronto, Ont., Canada, Canada, 2012. Canadian Information Processing Society.

[19] Fulvio Domini and Corrado Caudek. 3-d structure perceived from dynamic information: a new theory. *Trends in Cognitive Sciences*, 7(10):444–449, 2003.

[20] Mira Dontcheva, Gary Yngve, and Zoran Popović. Layered acting for character animation. *ACM Trans. Graph.*, 22(3):409–416, July 2003.

[21] Christopher K. Eveland, Diego A. Socolinsky, and Lawrence B. Wolff. Tracking human faces in infrared video. *Image Vision Comput.*, 21(7):579–590, 2003.

[22] S. H. Ferris. Motion parallax and absolute distance. *Journal of Experimental Psychology*, 95(2):258–263, 1972.

[23] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[24] George W. Fitzmaurice, Hiroshi Ishii, and William A. S. Buxton. Bricks: laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 442–449, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[25] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, I3D '97, pages 139–ff., New York, NY, USA, 1997. ACM.

[26] David Goldstein, Diane Haldane, and Carolyn Mitchell. Sex differences in visual-spatial ability: The role of performance factors. *Memory & Cognition*, 18(5):546–550, 1990.

[27] Ankit Gupta, Dieter Fox, Brian Curless, and Michael Cohen. Duplotrack: a real-time system for authoring and guiding duplo block assembly. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 389–402, New York, NY, USA, 2012. ACM.

[28] C. Heindl and C. Kopf. Reconstructme. http://reconstructme.net.

[29] Julie Heiser, Doantam Phan, Maneesh Agrawala, Barbara Tversky, and Pat Hanrahan. Identification and validation of cognitive design principles for automated generation of assembly instructions. In *Proc. AVI '04*, pages 311–319. ACM Press, 2004.

[30] Robert Held, Ankit Gupta, Brian Curless, and Maneesh Agrawala. 3d-puppetry: a kinect-based interface for 3d animation. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 423–434, New York, NY, USA, 2012. ACM.

[31] Steven J. Henderson and Steven K. Feiner. Augmented reality in the psychomotor phase of a procedural task. In *Proc. ISMAR '11*, pages 191–200. IEEE Computer Society, 2011.

[32] Knud Henriksen, Jon Sporring, and Kasper Hornbaek. Virtual trackballs revisited. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):206–216, March 2004.

[33] Lei Hou and Xiangyu Wang. Using augmented reality to cognitively facilitate product assembly process. *Augmented Reality*, pages 99–112, 2010.

[34] Hiroyasu Ichida, Yuichi Itoh, Yoshifumi Kitamura, and Fumio Kishino. Interactive retrieval of 3d shape models using physical objects. In *Proceedings of the 12th annual ACM international conference on Multimedia*, MULTIMEDIA '04, pages 692–699, New York, NY, USA, 2004. ACM.

[35] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. UIST '11*, pages 559–568. ACM Press, 2011.

[36] Jakob Sprogoe Jakobsen, Jesper Martin Ernstvang, Ole Juul Kristensen, and Jacob Allerelli. Automatic generation of building instructions for building element models. Patent publication number US7979251 B2, July 2011.

[37] Michael Patrick Johnson, Andrew Wilson, Bruce Blumberg, Christopher Kline, and Aaron Bobick. Sympathetic interfaces: using a plush toy to direct synthetic characters. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '99, pages 152–158, New York, NY, USA, 1999. ACM.

[38] Ricardo Jota and Hrvoje Benko. Constructing virtual 3d models with physical building blocks. In *Proc. CHI EA '11*, pages 2173–2178. ACM Press, 2011.

[39] Wendy Ju, Leonardo Bonanni, Richard Fletcher, Rebecca Hurwitz, Tilke Judd, Rehmi Post, Matthew Reynolds, and Jennifer Yoon. Origami desk: integrating technological innovation and human-centric design. In *Proc. DIS '02*, pages 399–405. ACM Press, 2002.

[40] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recogn.*, 40(3):1106–1122, March 2007.

[41] Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Joseph J. LaViola, Jr. Cavepainting: a fully immersive 3d artistic medium and interactive experience. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 85–93, New York, NY, USA, 2001. ACM.

[42] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[43] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 214–224, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[44] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[45] Robert E. Kraut, Susan R. Fussell, and Jane Siegel. Visual information as a conversational resource in collaborative physical tasks. *Hum.-Comput. Interact.*, 18(1):13–49, June 2003.

[46] Akinobu Lee, Tatsuya Kawahara, and Kiyohiro Shikano. Julius - an open source real-time large vocabulary recognition engine. In Paul Dalsgaard, Brge Lindberg, Henrik Benner, and Zheng-Hua Tan, editors, *Interspeech*, pages 1691–1694. ISCA, 2001.

[47] Gun A. Lee, Gerard J. Kim, and Mark Billinghurst. Immersive authoring: What you experience is what you get (wyxiwyg). *Commun. ACM*, 48(7):76–81, July 2005.

[48] Yong Jae Lee, C. Lawrence Zitnick, and Michael F. Cohen. Shadowdraw: real-time user guidance for freehand drawing. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 27:1–27:10, New York, NY, USA, 2011. ACM.

[49] Wilmot Li, Maneesh Agrawala, Brian Curless, and David Salesin. Automated generation of interactive 3d exploded view diagrams. In *Proc. SIGGRAPH '08*, pages 101:1–101:7. ACM Press, 2008.

[50] V. LoBrutto. Stanley kubrick: a biography. page page 430. Da Capo Press, Incorporated, 1999.

[51] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[52] Andrew Miller, Brandyn White, Emiko Charbonneau, Zach Kanzler, and Joseph J. LaViola Jr. Interactive 3d model acquisition and tracking of building block structures. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):651–659, April 2012.

[53] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. In *Proc. SIGGRAPH '10*, pages 58:1–58:12. ACM Press, 2010.

[54] Jose M. Molineros. *Computer vision and augmented reality for guiding assembly*. PhD thesis, Pennsylvania State University, University Park, PA, USA, 2002.

[55] Darnell J. Moore, Roy Want, Beverly L. Harrison, Anuj Gujar, and Ken Fishkin. Implementing phicons: combining computer vision with infrared technology for interactive physical icons. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, UIST '99, pages 67–68, New York, NY, USA, 1999. ACM.

[56] Christian Müller-Tomfelde. *Tabletops - Horizontal Interactive Displays*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[57] Cory S. Myers and Lawrence R. Rabiner. Comparative Study of Several Dynamic Time-Warping Algorithms for Connected-Word Recognition. *The Bell System Technical Journal*, 60(7), 1981.

[58] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads programming*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996.

[59] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 313–318, New York, NY, USA, 2003. ACM.

[60] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proc. UIST '11*, pages 135–144. ACM Press, 2011.

[61] David D. Preiss and Robert J. Sternberg. Effects of technology on verbal and visual-spatial abilities. *Cognitive Technology*, 11(1):14–22, 2006.

[62] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[63] David B. Reister. The least squares fit of a hyperplane to uncertain data. *Robotica*, 15(4):461–464, July 1997.

[64] Lincoln Ritter, Wilmot Li, Brian Curless, Maneesh Agrawala, and David Salesin. Painting with texture. In *Proceedings of the 17th Eurographics conference on Rendering Techniques*, EGSR'06, pages 371–376, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[65] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *Proc. ICCV '98*, pages 59–66. IEEE Computer Society, 1998.

[66] Jan Ruegg, Oliver Wang, Aljoscha Smolic, Markus Gross, Thomas Auzinger, Michael Guthe, and Stefan Jeschke. Ducttake: Spatiotemporal video compositing. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2013)*, 32(2), May 2013.

[67] Radu B. Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4. IEEE, May 2011.

[68] Kimiko Ryokai, Stefan Marti, and Hiroshi Ishii. I/o brush: drawing with everyday objects as ink. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 303–310, New York, NY, USA, 2004. ACM.

[69] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *In Advances in Neural Information Processing Systems 17*, pages 1185–1192, 2004.

[70] Mark Segal and Kurt Akeley. The OpenGL Graphics System: A Specification. Technical report, Silicon Graphics Inc., December 2006.

[71] Jia Sheng, Ravin Balakrishnan, and Karan Singh. An interface for virtual 3d sculpting via physical proxy. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, GRAPHITE '06, pages 213–220, New York, NY, USA, 2006. ACM.

[72] R. N. Shepard and J. Metzler. Mental rotation of three-dimensional objects. *Science*, 171:701–703, 1971.

[73] HyoJong Shin and Takeo Igarashi. Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In *Proceedings of Graphics Interface 2007*, GI '07, pages 63–70, New York, NY, USA, 2007. ACM.

[74] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 245–254, New York, NY, USA, 1985. ACM.

[75] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1297–1304, Washington, DC, USA, 2011. IEEE Computer Society.

[76] Heinrich Stumpf and John Eliot. A structural analysis of visual spatial ability in academically talented students. *Learning and Individual Differences*, 11(2):137–151, 1999.

[77] L.M. Surhone, M.T. Timpledon, and S.F. Marseken. *Lego Digital Designer*. VDM Verlag Dr. Mueller AG & Company Kg, 2010.

[78] Arthur Tang, Charles Owen, Frank Biocca, and Weimin Mou. Comparative effectiveness of augmented reality in object assembly. In *Proc. CHI '03*, pages 73–80. ACM Press, 2003.

[79] F. Thomas and O. Johnston. *Disney Animation: The Illusion of Life*. Abbeville Press, 1987.

[80] A tool developed with the support of the 3D-CoForm project. Meshlab. http://meshlab.souceforge.net.

[81] Brygg Ullmer and Hiroshi Ishii. The metadesk: models and prototypes for tangible user interfaces. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, pages 223–232, New York, NY, USA, 1997. ACM.

[82] H. Wallach and D. N. O'Connell. The kinetic depth effect. *Journal of Experimental Psychology*, 45(4), 1953.

[83] Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging physical and virtual worlds with electronic tags. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '99, pages 370–377, New York, NY, USA, 1999. ACM.

[84] Frank Weichert, Daniel Bachmann, Bartholomus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.

[85] Mingquan Zhou, Qingsong Huo, Guohua Geng, and Xiaojing Liu. A new 3d model retrieval method with building blocks. *Int. J. Comput. Games Technol.*, 2009:2:1–2:6, January 2009.

VITA

Ankit Gupta grew up in Delhi, India and received his B.Tech. in Computer Science & Engineering from Indian Institute of Technology (IIT), Delhi in 2007. He then joined the Department of Computer Science and Engineering at University of Washington for graduate studies, where he worked primarily with Prof. Brian Curless and Dr. Michael Cohen. He received his M.S. in Computer Science from UW in June 2009 and then his Ph.D. in August 2013.