DuploTrack: A Real-time System for Authoring and Guiding Duplo Block Assembly

Ankit Gupta, Dieter Fox, Brian Curless
University of Washington
Seattle, WA, USA
{ankit,fox,curless}@cs.washington.edu

Michael Cohen
Microsoft Research
Redmond, WA, USA
mcohen@microsoft.com

ABSTRACT

We demonstrate a realtime system which infers and tracks the assembly process of a snap-together block model using a Kinect® sensor. The inference enables us to build a virtual replica of the model at every step. Tracking enables us to provide context specific visual feedback on a screen by augmenting the rendered virtual model aligned with the physical model. The system allows users to author a new model and uses the inferred assembly process to guide its recreation by others. We propose a novel way of assembly guidance where the next block to be added is rendered in blinking mode with the tracked virtual model on screen. The system is also able to detect any mistakes made and helps correct them by providing appropriate feedback. We focus on assemblies of Duplo® blocks.

We discuss the shortcomings of existing methods of guidance — static figures or recorded videos — and demonstrate how our method avoids those shortcomings. We also report on a user study to compare our system with standard figure-based guidance methods found in user manuals. The results of the user study suggest that our method is able to aid users' structural perception of the model better, leads to fewer assembly errors, and reduces model construction time.

Author Keywords

Active visual feedback; Depth camera; Assembly tasks; Tracking; Augmented reality; Virtual reality.

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: User Interfaces.; H.5.1. Information Interfaces and Presentation: Multimedia Information Systems.; I.2.10. Artificial Intelligence: Vision and Scene Understanding.; I.3.6. Computer Graphics: Methodology and Techniques.; I.4.8. Image Processing and Computer Vision: Scene Analysis.

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.

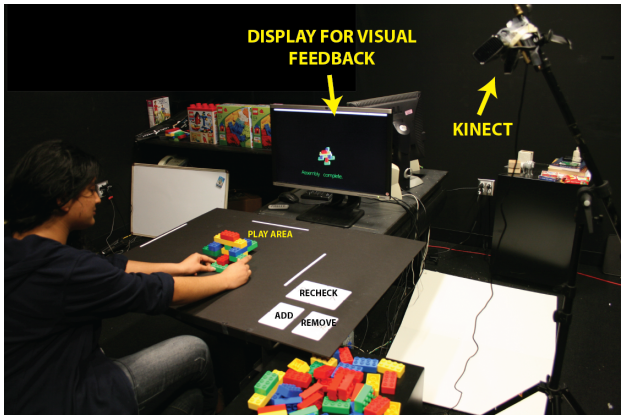


Figure 1: System setup. The user builds the model in the *Play area* and uses the *Add/Remove/Recheck boxes* to interact with the system. The Kinect® looks down from the right and passes the captured video/depth stream to our system to track the model and infer the assembly process in realtime. Visual feedback is shown on the display in front of the user.

Model assembly often follows printed step-by-step instructions as shown in Figure 3. 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 [26] 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 fol-

UIST'12, October 7–10, 2012, Cambridge, Massachusetts, USA.

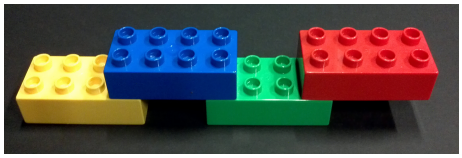


Figure 2: Our system uses 2×4 Duplo® blocks.

low especially if there is any back-tracking in the construction process. A user could also create a well-annotated set of instructions in 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.

In this paper, we design and implement a system which watches the user assemble a model using a camera with depth sensing and infers the assembly process in realtime. Figure 1 shows the setup of our system. It dynamically tracks the physical model 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.

The Authoring and Guidance system uses a combination of color-based and depth-based (3D) tracking. We leverage the widely available Kinect® sensor which gives us depth information along with color. Although blocks come in varying shapes and sizes, to keep the inference problem computationally tractable, we work here only with Duplo® blocks that each have a 2×4 arrangement of studs, as shown in Figure 2. We later discuss how our solution can be extended to include other different types of blocks. Even with both color and depth sensors and the single block type, inconsistent and noisy depth sensing, specular reflections, and the partial occlusions caused by the user's hands present significant challenges to creating a responsive and robust system. We counter all these challenges in our work both through sensor fusion, and by designing the user interface to lead the user gracefully through the block modeling process. We present a working system which allows for realtime inference and feedback in block model assemblies.

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

We explore two broad directions in this paper - a novel way of guiding assembly process and a realtime system to track

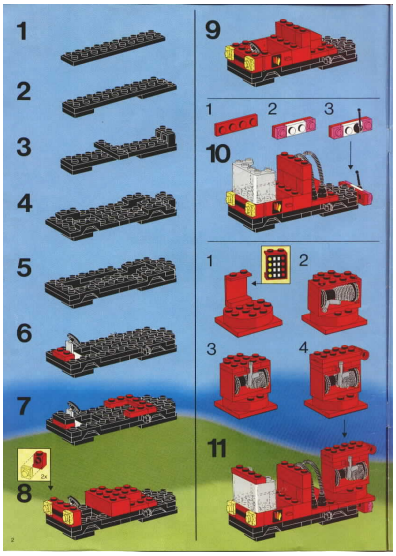


Figure 3: Traditional method of guiding model assembly tasks using images.

and infer the assembly process. In the next two sections, we review the prior work in these directions. We then present the design, implementation and potential applications of our system. After that, we describe the user study, analyze its results and conclude the paper by discussing potential future work.

RELATED WORK: GUIDING THE ASSEMBLY

Agrawala et al. [1] provide an excellent summary of the issues in designing the presentation of many assembly tasks. Heiser et al. [10] and Agrawala et al. [2] 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.

We 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. 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 [7] has shown that motion cues also play a central role in understanding shape. Motion parallax [8] and the kinetic depth effect [28] 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. We call this perceptual alignment from virtual model to physical model *perception transfer*. Perception transfer can induce errors in structural understanding which we 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. [21] have developed a system for generating Pause-and-Play video tutorials and discuss these common problems.

Kraut et al. [16] 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. Our system is related to this work in the sense that we provide 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 [9, 25, 22, 6] 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. [27], Henderson et al. [11] and Boud et al. [5], which evaluate and demonstrate the advantages of using AR techniques over the traditional figure-based techniques. Hou et al. [12] 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, the models are typically stationary and the motion cues are due to parallax caused by head motion. In our system, we 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 our system. One of our 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.

We experimentally compare our system with the figure-based guidance method that is most commonly used. We do not compare with the recorded video-based systems because, in essence, our system 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.

RELATED WORK: TRACKING THE ASSEMBLY

There has also been work to track and evaluate the correctness of assembly steps presented to the user. Molineros et al. [20] 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. [15] 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. Our work presents a realtime system which 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. [14] 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. [3] 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. Our system 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. [18] 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 ac-

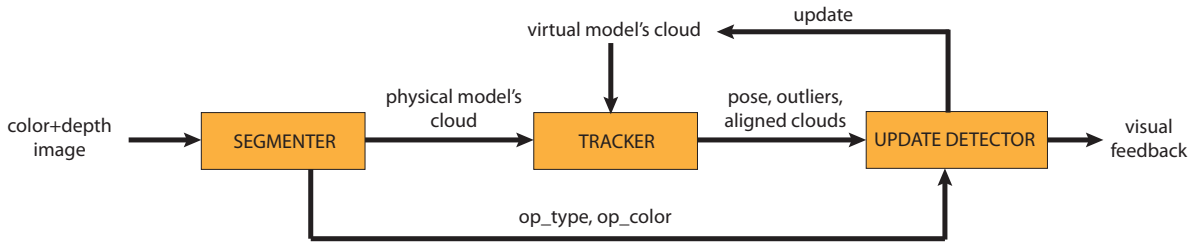


Figure 4: Processing pipeline: The Segmenter receives a color plus depth image from the camera and segments out the background and user's hands. The remaining image is passed as a 3D point cloud to the Tracker which aligns the virtual model's point cloud with that from the camera to output the pose. The Segmenter also detects the user's choice of operation and passes this to the Update Detector as operation type and color of block being added or removed. The Update Detector takes this information and the tracked pose to detect any update. Appropriate visual feedback is rendered to the user and the virtual model is updated.

curacy shows scanning errors due to tracking misalignment, hand pixels, and parts that are less visible. In comparison, our work 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, we enable two-way feedback between the user and the system compared to only system-to-user feedback in their work. We believe this leads to richer human-computer interaction experience. Additionally, we report on a user study comparing motion-tracked guidance to traditional methods.

SYSTEM OVERVIEW

Our system consists of a Kinect® depth+color camera above and to the right of a table surface in front of which the user sits as shown in Figure 1. The Kinect® looks obliquely down on the table surface. At the back of the table is a monitor facing the user; when the physical model is being tracked, the monitor shows a virtual replica of the model from the point of view of the user. A set of 2×4 Duplo® blocks sits off to the side of the table within reach of the user. The table surface has four demarcated regions - *Play area*, *Add box*, *Remove box* and *Recheck box*. We refer to the last three together as *Control boxes*.

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. Please see the supplementary video for a recordings of system usage.

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

Representation for Block Models

We assume 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 Duplo® blocks either red, green, blue or yellow in color. Each block occupies a volume equivalent to 8 voxels.

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

- List of blocks where each block has a color 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.

We also have a mesh model and point cloud representation for one 2×4 block which can be added to or removed from the virtual model in any color. The mesh model is 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 this block point cloud, with points removed in areas that are covered by other blocks.

Processing Pipeline

Figure 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 *Segmenter* module takes each color+depth image from the camera and prunes it to the pixels corresponding to the physical model in the *Play area*. It also checks for user inputs through *Control boxes* and converts that into an *op_type* (add, remove or recheck) and *op_color* (for add or remove). The *Tracker* module aligns the model's point cloud with the virtual model's point cloud. This aligned data plus the user inputs are then passed on to the *Update Detector* module.

SEGMENTING THE CAMERA DATA

We first convert the Kinect® color+depth images into colored 3D *camera point clouds*, P_c, by back-projecting the pixels to the known depths from the Kinect®. We filter out pixels which do not correspond to the physical model such as the background and hands. The system starts with an empty work area and we store 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, we mark that pixel as foreground. The 95% value is empirically determined to counter the depth inaccuracies from the Kinect® camera. We also filter out the pixels whose corresponding 3D points do not lie within the *Play area* or any of the *Control* boxes.

Pixels above the *Control boxes* set the *op_type* and *op_color* based on majority occupancy. The *Recheck* box is given priority over *Add* and *Remove* boxes since it lies further from the user. If *op_type* is add or remove, we set *op_color* by finding the majority color of the points in the corresponding box.

The remaining 3D pixels above the *Play area* are either part of the user's hands or the model. We use a color-histogram-based classifier built prior to the start of the system to distinguish between the hands and Duplo® blocks, and remove the hand pixels. The remaining pixels above the play area form the point cloud corresponding to the physical model, P_c.

TRACKING THE MODEL

A key component of the system is the ability to track the physical model as the user moves it around with or without adding/removing a block. This allows us to render the virtual model from the user's viewpoint on the screen and enables us to render additional feedback or instructions about the next step of the task. The tracking system runs at real-time frame rates. Currently, we do not guarantee robust tracking if the model is turned upside down. This is because the sensor cannot capture enough geometry on the hollow side of blocks and hence leading to less reliable tracking.

For every camera point cloud, P_c, we compute a transformation that aligns the virtual cloud, P_v, with P_c. We use the ICP (Iterative Closest Point) algorithm proposed by Besl and McKay [4] to align the point clouds by solving for a 6D transformation: 3D rotation and 3D translation. As a side effect of the algorithm, we also get a set of outliers, O, points in P_c which do not find a match in P_v. When a user makes changes to the model, these "outliers" correspond to differences between the previous and updated state of the model; thus, we use the outliers later for update detection.

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 (for the purpose of alignment, though retained for model update). 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, we reduce the outlier-rejection threshold.

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 model first appears before the camera, there is no previous transformation available. Further, the system can lose track of the physical model, 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*.

Pose Initialization

To initialize the pose, we need to estimate a suitable 3D translation and 3D rotation. For 3D translation, we simply use the difference between the 3D centroids of P_v and P_c.

To estimate an initial rotation, we exploit the fact that the model primarily consists of mutually orthogonal or parallel planar faces. We 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.

First, we estimate normals at every point in P_c based on each point's position and the positions of its nearest neighbors. We cluster the normals 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 has the most normals supporting it. 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. We prune out half of the possible orientations by assuming the hollow side of the blocks does not face up. Hence the 3D rotation search space can be pruned to testing 12 candidate orientations, $j = 1...12$.

Then, for each orientation candidate j, we transform the virtual model points P_v 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. We then run the ICP algorithm 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®. We choose the best transformation candidate T_j

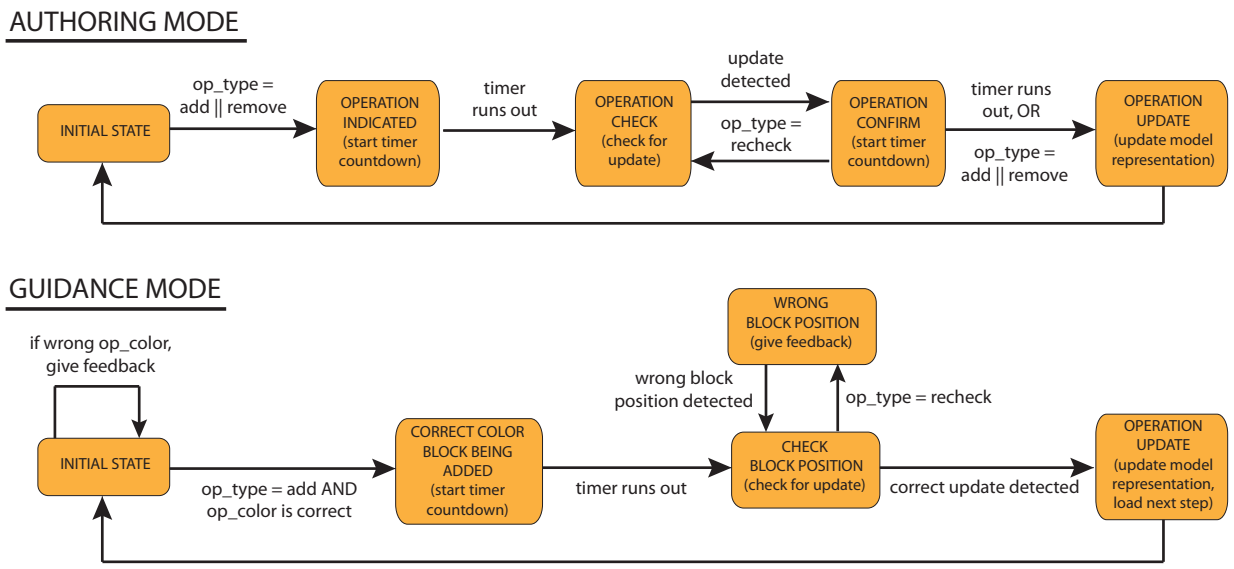


Figure 5: The Update Detector is implemented as a finite state machine with user input and tracked data as input to update the model. In Authoring mode, it detects an update and applies it unless a recheck is requested by the user. In Guidance mode, it detects and performs the update and displays feedback to the user if there is a mistake.

which has the minimum match error between P_c and P_j. For the match error, we render the colored point clouds P_c and transformed P_j from the camera's viewpoint to create two color images. We first quantize the image colors into the possible red, green, blue, and yellow model colors and partition the images into abutting 50×50 windows. For each window, we build a color histogram and cluster 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. We put them in a separate bin called saturated-color bin. We set the match error between the images as the sum of the match costs between the histograms of corresponding windows.

We use the Earth Mover's Distance (EMD) [23] 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. We set the cost to move pixels from any color to a different color as 1. We set the cost to move pixels from saturated-color bin to any color bin as 0.

Model Tracking at the Start

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 we ask the user 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. We modify the matching algorithms for aligning point clouds to allow for small translations from the known pose. Once the model has 5 blocks, we remove this restriction and allow the user to freely move the model around. The system now dynamically tracks it as described earlier.

DETECTING MODEL UPDATES

When the user indicates an update to the model (add or remove), the system uses the output of the tracker to detect the update. Separate finite state machines implement the Authoring and Guidance modes as shown in Figure 5. These are a formal representation of the system usage already described in detail in the System Outline section.

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.

Creating the Candidate Set for Updates

At any stage in the assembly we maintain 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 addition mode, we traverse the voxel space and find sets of unoccupied voxels where a new block could be added. To ensure connectivity in the model, these sets must be connected to at least two occupied voxels; i.e., we 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. We can do this analysis using the voxelized representation to find such candidates.

We also add one more candidate 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, we 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 tracked camera data stream consisting of (1) the point cloud seen by the camera, P_c, (2) the transformation T which best aligns the un-updated virtual model's point cloud, P_v, to P_c and (3) the outlier points, O, given T.

Updating the Belief Distribution

After each new block addition or removal, all the $b(j)$ are set to 0. We then update the $b(j)$'s 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, we accumulate beliefs from more than one pose of the model as the user turns the model to reveal new views.

From each pose, we score the possible candidates 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. For a removal candidate, P_j is obtained by removing the point cloud of the block from P_v. For the no addition/removal candidate, P_j is simply equal to P_v.

To compute the matching score, we first align P_c with each of the P_j's using ICP. The P_j's are structurally close to P_v and we have already aligned P_v with P_c using the transformation T. Hence we use T as the initial transform for these ICP alignments. We then compute the match error for each candidate using the same matching metric as used for comparing poses for tracking . For addition candidates, we also set the error to infinite if the majority of outliers which occupy the voxels of the candidate are not of the indicated color. We do the same for removal candidates if the corresponding block is not of the indicated color.

We sort the match errors 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, we require 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, we remove this requirement). To accumulate beliefs from three well-separated views, we begin 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. We 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, we normalize each set to sum to 1 and then sum all normalized sets 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 we declare the highest scoring candidate as the winner. If we cannot declare a winner, then we wait 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, we reset the belief sets and re-start checking for an update.

Moving to the Next Step

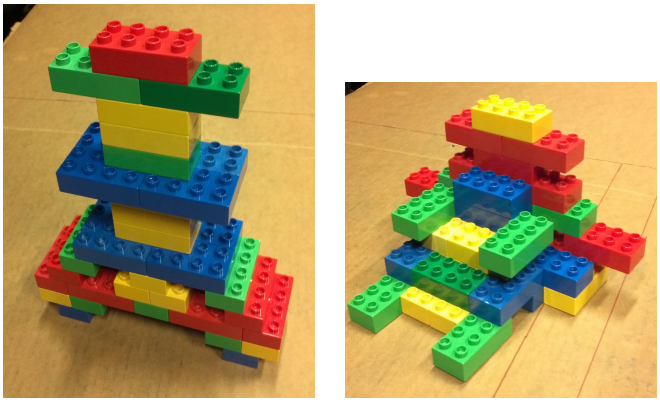
If the system selects a candidate as the update but the user then asks for a recheck, we remove that candidate from further consideration, reset the belief sets, and the update checking process restarts. In practice, we find that the user needs to request a recheck less than one out of twenty additions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

Once the model update has been identified (and approved by the user in Authoring mode), we update the internal representation of the model. The model's virtual cloud P_v is set to the point cloud corresponding to the winning candidate. For block addition, we append the new block 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.

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



(a) (b)

Figure 6: Models authored using our system.

if the model motion is fast. We have used our system to build models up to 85 blocks in size. At that point the tracking speed reduces to about 5 frames per second. Figure 6 shows a few models that users have authored while using our system.

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 we were limited by the number of CPU threads that we could use. In the future we believe that our system can benefit from exploiting the parallelization potential in GPUs as shown by recent works like *KinectFusion* [13].

We now describe a few applications of our system.

Generating Tutorials for Model Assembly

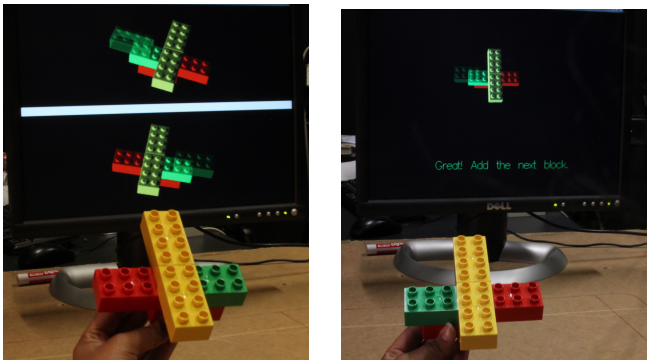
The captured assembly of a model is a sequence of add or remove operations. To generate an assembly tutorial, we would like to omit any steps that involve adding and then later removing a block. We delete such add/remove operations from the representation. This may cause the remainder of the sequence to have block additions which do not connect to any blocks added before them. We reorder the steps by postponing the addition of such blocks until they have a connection with the prior model. It is easy to prove that the assembly sequence of any model can be reordered in this way. This representation can then be used to then generate tutorials or explanations in form of images with or without annotations [17, 19].

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.

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.



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

Figure 7: Photographs of the two guidance interfaces that we 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.

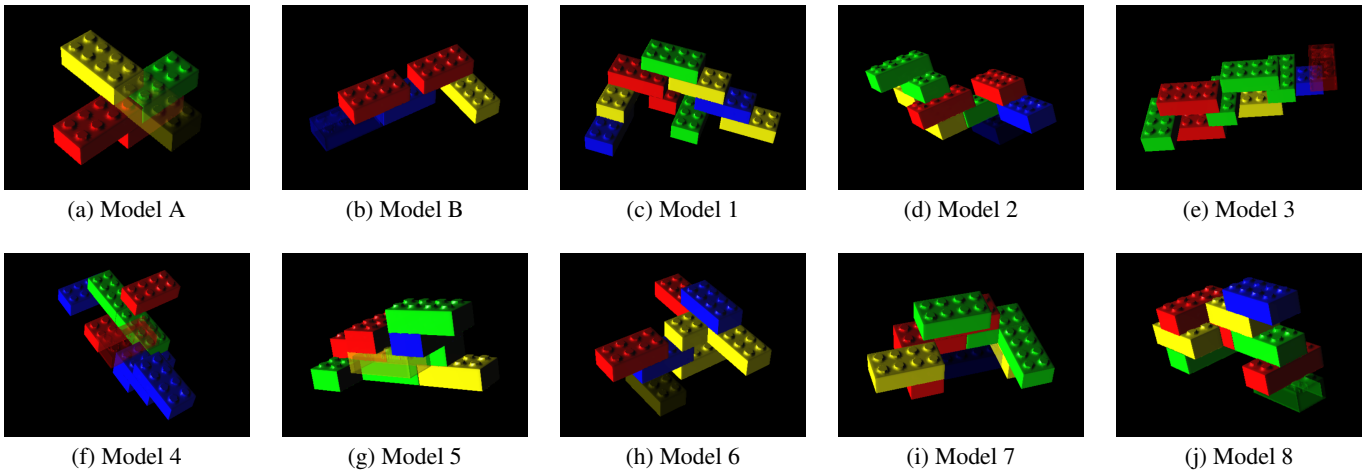
USER STUDY

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, we conducted a study that focuses on the Guidance system as it can be compared to other more traditional guidance modes.

In particular, we 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 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 our 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 mode as in *Baseline*. Figure 7b shows a screenshot of this interface. Please refer to supplementary videos which show how participants use these interfaces.

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

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



(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 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.

Task Design

We assembled ten initial block models, shown in Figure 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. We 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, we measured the time taken to add that block. We also noted if the block addition was correct or not. We analyze the effect of the interface conditions on these two dependent variables - the time taken to make the update and correctness.

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 [24] 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 we wanted to test this possible dependence. We 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 our analysis, we 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, we 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 we did not want the effect of a mistake at an earlier stage to cause a mistake at a later stage. We did not record the time taken for corrective feedback and correction in our time record of the task completion.

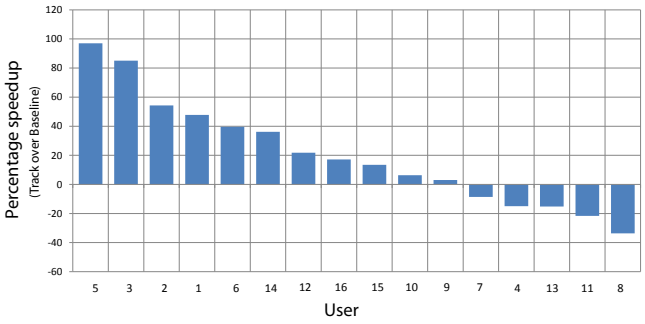
After the participants finished all the tasks, they were asked to fill a post-study survey which asked for qualitative feedback about the two conditions. We asked them about their preference for the systems, if any, and also for any other interface ideas that could help guide them better. We report qualitative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

Participants

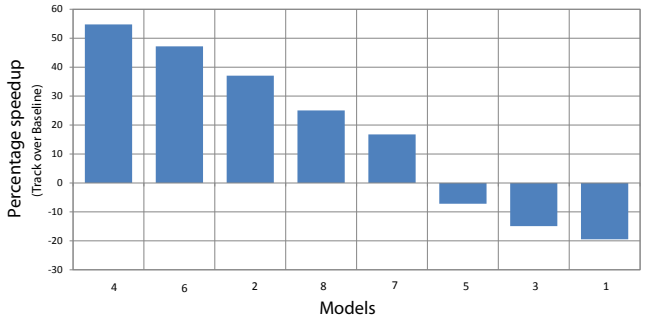
Sixteen participants (eight female, eight male, ages 20 to 30) volunteered. Twelve of them had built models with blocks as a child and all of them had experience in doing some 3D assembly tasks like furniture, electronics etc. Each participant's study lasted for about 45 minutes.

Results

We analyze the results of the experiment in three ways. First, we 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, we 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. We analyze each of the two models separately for this. Third and last, we report the qualitative feedback from our participants about the two interfaces.



(a) Percentage speedup for users.



(b) Percentage speedup for models.

Figure 9: Percentage speedup over all the one-block addition tasks (value greater than 0 means that the *Track* interface takes less time).

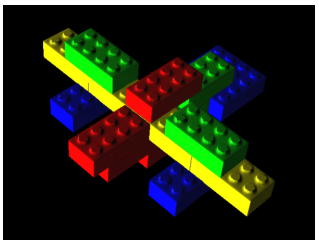
Comparing the Interfaces for the One Block Additions

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

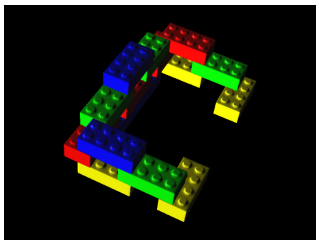
We perform a mixed-model analysis of variance where *Time* is the dependent factor, and *Participant* and *Model* are random effects. Modeling *Participant* as random effect accounts for any variation in individual performance, and modeling *Model* accounts for any difference in difficulty of block addition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



(a) Model A.



(b) Model B.

Figure 10: The two 16-block models, each built by adding 12 blocks sequentially by the participants.

times is from 0.189 seconds to 5.687 seconds of improvement for *Track* over *Baseline*.

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

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

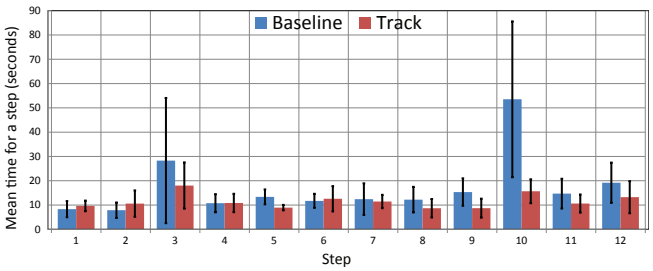
A value above zero indicates that *Track* took less time and vice versa. In Figures 9a and 9b we show the *speedup* for individual users and models respectively. Although the per-user and per-model time averages have been aggregated over a small sample of the models and users respectively, they provide 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*. We 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.

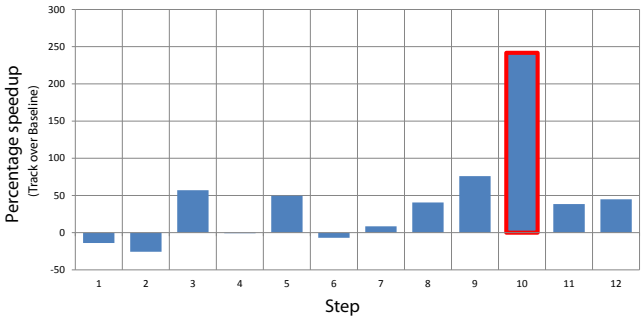
Comparing the Interfaces for Multi-Block Additions

Here, we analyze the times to add blocks one after the other to the same model. We 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, we consider two 4-block models to which the participants make 12 sequential block additions using the two interfaces, one interface per model. We do a mixed-model analysis of variance for each of the models separately. Figure 10 shows the two complete models and we refer to them as Model A and Model B. As before, we use *Participant* as a random effect and *Interface* as a fixed effect. We add the size of the model (number of blocks) at every step, denoted by variable *Step*, as another fixed effect. In this analysis, we 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

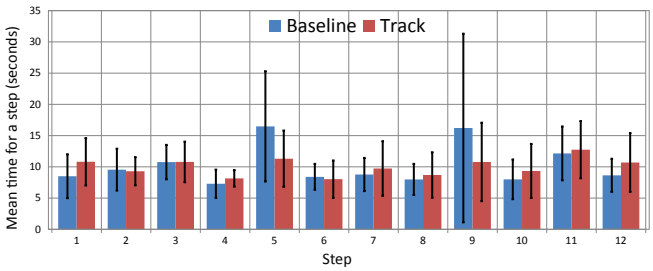


(a) Mean time taken per block addition and standard deviation (seconds). Less is better.

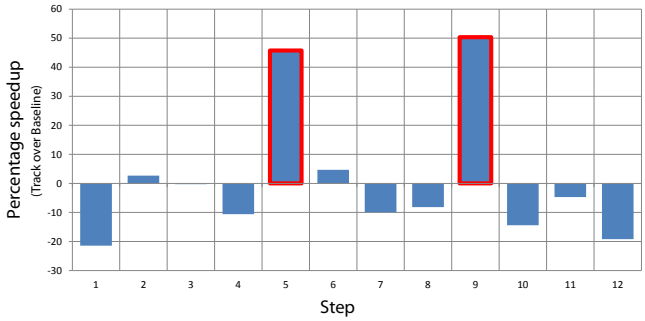


(b) Percentage speedup (value greater than 0 means that the *Track* interface takes less time. The red outline indicates statistically significant result ($p < 0.05$))

Figure 11: Metrics for Model A.



(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 12: Metrics for Model B.

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 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 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 12a shows the mean times for each step. The *Interface* by *Step* interaction does show significant improvement of *Track* in some cases. Figure 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, we 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 13 shows this scatter diagram. We 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 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".

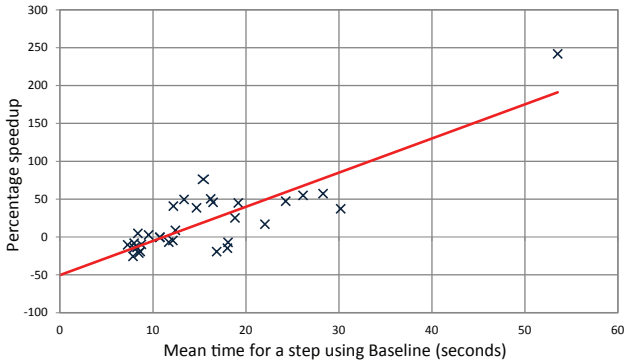


Figure 13: Scatter plot of percentage speedup vs mean time taken to do a block addition using *Baseline* (an indicator of the step's hardness). Fitting a line using least squares (shown in red) indicates a positive correlation between the mean time and speedup obtained by using *Track* over *Baseline*.

Based on our interactions with the users, we 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. We observed no mistakes made with the *Track* system while a total of 10 mistakes were made with the static *Baseline* instructions. We also observed significant speedups for the tasks with the dynamic systems.

Clearly, we have only scratched the surface of the questions raised. All of the models are quite simple compared to the richness of Lego® models available. We can conjecture that our results may even be stronger as the complexity increases but we 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. We can hope that this aspect of the system cam improve over time.

CONCLUSION AND FUTURE WORK

We have demonstrated a system which tracks an evolving Duplo® block model in realtime. 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. We discuss the shortcomings of existing static figures or videos of the instruction steps and show how our guidance method avoids these. A user study comparing our system with the traditional figure-based guidance method suggests that our 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 our informal discussions with the participants, we 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, we 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.

We 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 are considering 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.

We 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. In fact, we have already extended our work to initially load a library of parts which the user wants to use in the model. The system uses a learnt area-based descriptor to distinguish parts. However, in the future we would like to handle blocks of curved shapes and articulated parts. It would also be useful to move away from a block-at-a-time update approach and allow for making subassemblies and merging them. This might require a different model representation and present more computational challenges.

We 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 we 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 and we plan to exploit this in future. We hope this initial work inspires others to take on new applications for this type of technology.

ACKNOWLEDGMENTS

We thank the reviewers for their insightful comments. This work was supported by funding from the University of Washington Animation Research Labs, Intel Science and Technology Center for Visual Computing, Microsoft, Adobe, and Pixar.

REFERENCES

1. Agrawala, M., Li, W., and Berthouzoz, F. Design principles for visual communication. *Communications of the ACM* (Apr. 2011), 60–69.
2. Agrawala, M., Phan, D., Heiser, J., Haymaker, J., Klingner, J., Hanrahan, P., and Tversky, B. Designing effective step-by-step assembly instructions. In *Proc. SIGGRAPH '03*, ACM Press (2003), 828–837.
3. Anderson, D., Frankel, J. L., Marks, J., Agarwala, A., Beardsley, P., Hodgins, J., Leigh, D., Ryall, K., Sullivan, E., and Yedidia, J. S. Tangible interaction + graphical interpretation: a new approach to 3d modeling. In *Proc. SIGGRAPH '00*, ACM Press/Addison-Wesley Publishing Co. (2000), 393–402.
4. Besl, P., and McKay, N. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14* (1992), 239–256.
5. Boud, A. C., Haniff, D. J., Baber, C., and Steiner, S. J. Virtual reality and augmented reality as a training tool for assembly tasks. In *Proc. IV '99*, IEEE Computer Society (1999), 32–36.
6. Casey, M. B., Winner, E., Brabeck, M. M., Sullivan, K., Gilhooly, K. J., Keane, M. T. G., Logie, R. H., and Erdos, G. 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* (1990), 275–294.
7. Domini, F., and Caudek, C. 3-d structure perceived from dynamic information: a new theory. *Trends in Cognitive Sciences 7*, 10 (2003), 444–449.
8. Ferris, S. H. Motion parallax and absolute distance. *Journal of Experimental Psychology 95*, 2 (1972), 258–263.
9. Goldstein, D., Haldane, D., and Mitchell, C. Sex differences in visual-spatial ability: The role of performance factors. *Memory & Cognition 18*, 5 (1990), 546–550.
10. Heiser, J., Phan, D., Agrawala, M., Tversky, B., and Hanrahan, P. Identification and validation of cognitive design principles for automated generation of assembly instructions. In *Proc. AVI '04*, ACM Press (2004), 311–319.
11. Henderson, S. J., and Feiner, S. K. Augmented reality in the psychomotor phase of a procedural task. In *Proc. ISMAR '11*, IEEE Computer Society (2011), 191–200.
12. Hou, L., and Wang, X. Using augmented reality to cognitively facilitate product assembly process. *Augmented Reality* (2010), 99–112.
13. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. UIST '11*, ACM Press (2011), 559–568.
14. Jota, R., and Benko, H. Constructing virtual 3d models with physical building blocks. In *Proc. CHI EA '11*, ACM Press (2011), 2173–2178.
15. Ju, W., Bonanni, L., Fletcher, R., Hurwitz, R., Judd, T., Post, R., Reynolds, M., and Yoon, J. Origami desk: integrating technological innovation and human-centric design. In *Proc. DIS '02*, ACM Press (2002), 399–405.
16. Kraut, R. E., Fussell, S. R., and Siegel, J. Visual information as a conversational resource in collaborative physical tasks. *Hum.-Comput. Interact. 18*, 1 (June 2003), 13–49.
17. Li, W., Agrawala, M., Curless, B., and Salesin, D. Automated generation of interactive 3d exploded view diagrams. In *Proc. SIGGRAPH '08*, ACM Press (2008), 101:1–101:7.
18. Miller, A., White, B., Charbonneau, E., Kanzler, Z., and LaViola Jr., J. J. Interactive 3d model acquisition and tracking of building block structures. *IEEE Transactions on Visualization and Computer Graphics 18*, 4 (Apr. 2012), 651–659.
19. Mitra, N. J., Yang, Y.-L., Yan, D.-M., Li, W., and Agrawala, M. Illustrating how mechanical assemblies work. In *Proc. SIGGRAPH '10*, ACM Press (2010), 58:1–58:12.
20. Molineros, J. M. *Computer vision and augmented reality for guiding assembly*. PhD thesis, Pennsylvania State University, University Park, PA, USA, 2002.
21. Pongnumkul, S., Dontcheva, M., Li, W., Wang, J., Bourdev, L., Avidan, S., and Cohen, M. F. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proc. UIST '11*, ACM Press (2011), 135–144.
22. Preiss, D. D., and Sternberg, R. J. Effects of technology on verbal and visual-spatial abilities. *Cognitive Technology 11*, 1 (2006), 14–22.
23. Rubner, Y., Tomasi, C., and Guibas, L. J. A metric for distributions with applications to image databases. In *Proc. ICCV '98*, IEEE Computer Society (1998), 59–66.
24. Shepard, R. N., and Metzler, J. Mental rotation of three-dimensional objects. *Science 171* (1971), 701–703.
25. Stumpf, H., and Eliot, J. A structural analysis of visual spatial ability in academically talented students. *Learning and Individual Differences 11*, 2 (1999), 137–151.
26. Surhone, L., Timpledon, M., and Marseken, S. *Lego Digital Designer*. VDM Verlag Dr. Mueller AG & Company Kg, 2010.
27. Tang, A., Owen, C., Biocca, F., and Mou, W. Comparative effectiveness of augmented reality in object assembly. In *Proc. CHI '03*, ACM Press (2003), 73–80.
28. Wallach, H., and O'Connell, D. N. The kinetic depth effect. *Journal of Experimental Psychology 45*, 4 (1953).