Interactive 3D Model Acquisition and Tracking of Building Block Structures

Andrew Miller, Brandyn White, Emiko Charbonneau, Zach Kanzler, and Joseph J. LaViola Jr., *Member, IEEE*

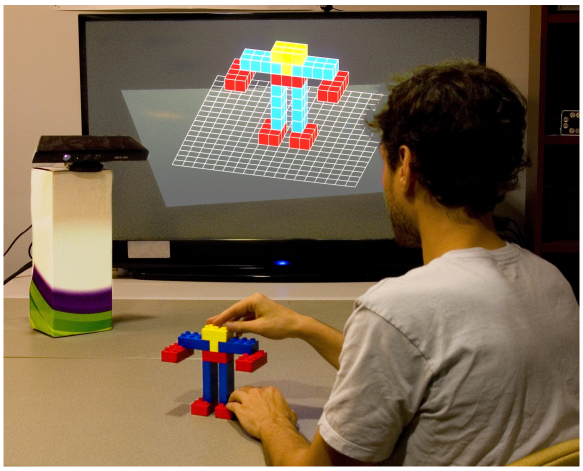

Fig. 1. An overhead view of our system in action. A depth camera observes the creation of a physical structure made of building blocks. Our system continuously updates a virtual model of the structure as the user adds and removes pieces.

Abstract—We present a prototype system for interactive construction and modification of 3D physical models using building blocks. Our system uses a depth sensing camera and a novel algorithm for acquiring and tracking the physical models. The algorithm, Lattice-First, is based on the fact that building block structures can be arranged in a 3D point lattice where the smallest block unit is a basis in which to derive all the pieces of the model. The algorithm also makes it possible for users to interact naturally with the physical model as it is acquired, using their bare hands to add and remove pieces. We present the details of our algorithm, along with examples of the models we can acquire using the interactive system. We also show the results of an experiment where participants modify a block structure in the absence of visual feedback. Finally, we discuss two proof-of-concept applications: a collaborative guided assembly system where one user is interactively guided to build a structure based on another user's design, and a game where the player must build a structure that matches an on-screen silhouette.

Index Terms—Interactive physical model building, 3D model acquisition, object tracking, depth cameras, building block structures.

1 Introduction

3D model acquisition, reconstruction, and tracking of physical objects have a long history in computer graphics, with applications in areas such as virtual and augmented reality, object modeling, simulation, CAD/CAM, and cultural heritage [6]. Although great strides have been made in this domain, highly interactive applications using this technology for virtual and augmented reality are still limited. Specifically, we envision a system where 3D models can be tracked in real time and acquired as they are being built. Such a system would support the interactive construction of 3D physical models that would have a virtual counterpart. Additionally, a projection system that accurately superimposes visual imagery on and surrounding the physical model would support natural user interface applications for collaborative assembly tasks and augmented reality.

To support our vision, there are three significant challenges that must be solved: (1) continuous tracking, acquisition, and reconstruction of 3D models as they evolve through time; (2) support for natural interaction where users manipulate and modify the models during the construction process with their bare hands; and (3) visual feedback that provides the user spatial information to assist in construction.

In this work, we focus on the first two of these challenges for the domain of building block structures. We use a novel algorithm, Lattice-First, that takes advantage of the orthogonal and grid-like properties of building block structures in order to achieve robustness to interference and occlusions from the user's hands, and to support a dynamic model in which pieces can be added and removed (see Figure 1). The algorithm is fast and effective, providing users with the ability to incrementally construct a block-based physical model while the system maintains the model's virtual representation. We provide the algorithm details, examples of models the algorithm can acquire, and a preliminary experiment that evaluates the effectiveness of our system with the added restriction of no feedback to the user. We also present two proof-of-concept applications that use a visual display to guide the user through physical construction tasks.

• *Andrew Miller is with University of Central Florida, E-mail: amiller@cs.ucf.edu.*
• *Brandyn White is with University of Maryland, E-mail: bwhite@cs.umd.edu.*
• *Emiko Charbonneau is with University of Central Florida, E-mail: miko@cs.ucf.edu.*
• *Zach Kanzler is with University of Central Florida, E-mail: othey4kman@gmail.com.*
• *Joseph J. LaViola Jr. is with University of Central Florida, E-mail: jjl@eecs.ucf.edu.*

Manuscript received 15 September 2011; accepted 3 January 2012; posted online 4 March 2012; mailed on 27 February 2012.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

2 Related Work

A significant amount of work has dealt with the problem of 3D model acquisition and reconstruction. The goal of a model acquisition system is to construct a virtual model of a physical object, using optical sensors such as cameras or laser scanners [9, 21].

Approaches to model acquisition typically consist of two high-level components: first, a registration algorithm that can estimate the camera poses of images from several viewpoints in order to align them to a common frame of reference, and second, a representation of the 3D object that supports fusing the information from the images. The most common representations are volumetric models (e.g., an occupancy grid [18]), or an implicit surface definition embedded in a distance field [8]). The registration algorithm typically is a variation of Iterative Closest Points (ICP) which involves computing a least-squares fit to estimate the camera pose [3, 33].

Although many of these systems allow the model to be acquired in realtime [15, 32, 41], they are not inherently robust enough to support natural user interaction during acquisition. In particular, the user's hands (and foreground objects generally) are often spuriously added to the model, subsequently causing alignment error and a positive feedback loop (i.e., *drift*) as discussed in Section 7. One approach is to segment out the pixels corresponding to the user's hands, although this typically requires additional constraints that impede natural user interaction, such as requiring the user to wear a colored glove [40], requiring a static background [34], or by relying on skin color segmentation [4]. Alternatively, KinectFusion supports real-time model acquisition of a scene *followed by* robust segmentation of foreground objects, but not both at the same time [15]. This is insufficient for the user-interaction scenario we consider, in which users construct and modify the physical object as it is being acquired.

Our approach involves introducing a domain restriction (i.e., orthogonal building blocks) that we can exploit to improve the robustness of tracking and acquisition. Previous work has similarly used geometric constraints to aid in 3D scene reconstruction [37]. Approaches that rely on the orthogonal features of buildings are commonly used in robotics and urban modeling [31]. A technique due to Schnabel uses RANSAC to identify shape primitives in a scene, such as cones, cylinders, and planes [35]. These bottom-up techniques could be applied as well to building block models; however, building block models satisfy a stronger constraint that we use: the block structures exhibit a global lattice-like symmetry and periodicity (see Section 3). The Complex Extended Gaussian Image (CEGI) is a method that decouples the translation and rotation components of the tracking problem in order to locate a prior model in a range image, even without an initial pose estimate [17]. Our method involves a similar decoupling (see Sections 4.2 and 4.3) and can be thought of as extending the CEGI to include the periodic structure of blocks, eliminating the need for the prior model. Thus we exploit the *a priori* knowledge of this regularity to improve the robustness of realtime acquisition.

There are several tangible interaction systems based on building blocks that involve instrumenting the blocks with electronics [1, 14, 19, 27, 36, 44]. Other construction toys have also been used, such as hub-and-spoke pieces [43]. In these systems, custom circuitry is used for accurate tracking and shape reconstruction, but at the cost of developing and making specialized parts. Such approaches to object tracking include inertial, acoustic, or magnetic sensing [42]. Another solution has been reported that requires a customized passive block without internal electronics [2]. An advantage of our depth camera system is that it can work with unmodified off-the-shelf building blocks.

We highlight several projects that explore the *physical* realization of initially *virtual* models. 'Popup' creates a printable papercraft template from a 3D model [22], and 'Plushie' generates fabric templates that can be sewn together to make a stuffed animal [24]. Using an projector-based augmented reality display, a 3D guide has been be projected onto a foam block to help the user carve out a complex shape with a wire tool [23]. We demonstrate the use of our system in this context with the collaborative guided assembly application described in Section 6.3.

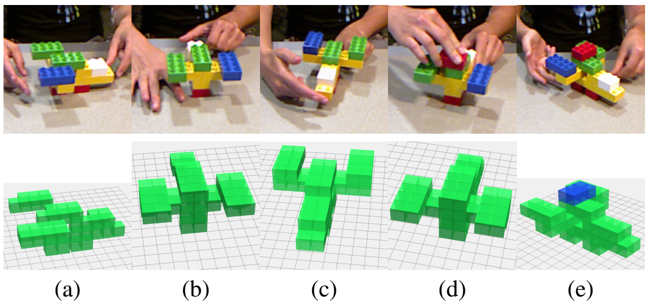

Fig. 2. Example of user interaction with our model acquisition system: (a) user places the structure in front of the sensor, and the virtual model is partially initialized; (b)(c) user rotates the structure to fill complete the model; (d) user adds a piece and then rotates the structure as (e) the model is updated with additional piece (rendered in blue).

Finally, we note that some 3D acquisition systems use color images instead of (or in addition to) depth images [7, 18, 25]. Our system uses only depth information to track and acquire the model since our future vision involves an application of projected augmented reality, in which the projected visible light would interfere with the color imagery. However, we optionally utilize color imagery in the final rendering step in order to display the correct color of the blocks (see Section 5).

3 Assumptions and Scope

This paper considers building block structures on a tabletop surface, which is a restricted subset of the general 3D model acquisition problem. Our algorithm takes advantage of these constraints in order to achieve robustness to interference and occlusion from the user's hands, and to support dynamic models where pieces can be added or removed (see Figure 2). Specifically, we consider building blocks arranged in a 3D point lattice where the smallest building block unit is the basis (see Figure 3). The lattice can be written as $\{N_1 w_x, N_2 w_y, N_3 w_z\}$ where $N_1, N_2, N_3 \in \mathbb{N}$ and w_x, w_y, w_z, are the dimensions of the smallest building block unit. For example, Duplo blocks have unit dimensions $(w_x, w_y, w_z) = (16mm, 19.2mm, 16mm)$. The physical interpretation of this lattice is that it contains all the possible corner points of the block structure. We assume that the block structure remains flat on the table, but can be slid around and rotated. The tracking problem is therefore constrained to a two-dimensional surface and has three degrees of freedom rather than six. Under these assumptions, points on the surfaces of the blocks lie on orthogonal planes intersecting the lattice and parallel to XY, YZ, or XZ. Our algorithm is intended to be general in respect to the shape and size of the building blocks used, however we assume that the blocks have a natural 'right-side-up' orientation and that the block unit width is the same in the X and Z axes (i.e., $w_x = w_z$). The Y axis is perpendicular to the table.

4 Lattice-First Algorithm

The Lattice-First algorithm first finds the transformation from *physical* coordinates to *model* coordinates, in which the blocks comprising the tracked object align with the coordinate system. This transformation is represented as the product of several matrices:

$$\mathbf{P}^{model} = (\mathbf{C})(\mathbf{T})(\mathbf{R})\mathbf{P} \tag{1}$$

where $\mathbf{P}$ is a 4xN matrix, N is the total number of points from the depth camera, $\mathbf{R}$ is a rotation matrix, $\mathbf{T}$ is a translation matrix, and $\mathbf{C}$ is a discrete correction (translation by a multiple of $w_x = w_z$ or rotation by a multiple of 90°) that aligns the current frame to a previous model estimate. After finding this transformation, the remaining goal is to update the voxel grid, estimating which voxels are occupied and which are vacant. The steps of the algorithm (see Figure 4) are summarized as follows:

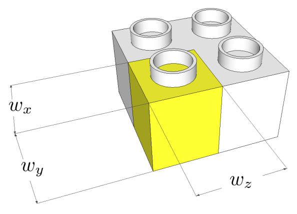

Fig. 3. Illustration of the smallest unit dimensions, w_x, w_y, and w_z, for the lattice-like building blocks discussed in this work. We assume that these dimensions are known parameters of the system.

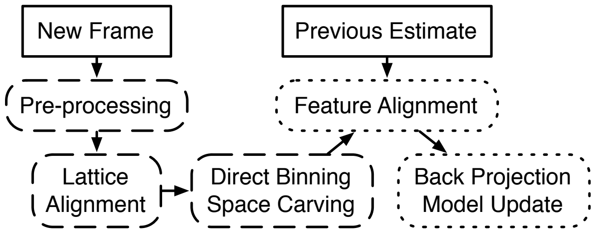

Fig. 4. High level flowchart of our proposed system. The processing stages up to Feature Alignment are computed directly for each frame, avoiding potential feedback loops with the dynamically updated model. Note that solid lines indicate data, dashed lines indicate processes computed directly for each frame, and dotted lines indicate processes that use both the current frame and the previous estimate.

1. (Section 4.1) Take a new depth image from the sensor, computing point positions $\mathbf{P}$ and surface normals $\hat{\mathbf{n}}$.

2. (Section 4.2) Use the surface normals $\hat{\mathbf{n}}$ to determine the lattice orientation $\mathbf{R}$ and the oriented point samples.

3. (Section 4.3) Determine the lattice translation (i.e., $\mathbf{T}$) and aligned point samples.

4. (Section 4.4) Bin the point samples to determine occupancy estimates and vacancy estimates.

5. (Section 4.5) Use space carving to augment the vacancy estimates.

6. (Section 4.6) Align the previous estimate to the current estimate using feature matching in a finite search space (i.e., find $\mathbf{C}$).

7. (Section 4.7) Backproject the union of the previous estimate and current estimate into the depth image to validate updates to the model.

4.1 Preprocessing

The input from the depth sensor is a Euclidean point cloud $\mathbf{P}$, arranged in a two-dimensional grid as a depth image. Calibration matrices that produce Euclidean measurements from depth sensor raw data may be provided by the manufacturer, or otherwise can be estimated experimentally [11]. We assume that the camera has also been extrinsically calibrated to the table surface so that the XZ plane of the measurements is coplanar with the table. This extrinsic calibration is performed by manually clicking four corner points in a depth image of the empty table surface. These four points are also used to define a 3D *volume of interest*, a prism extending upward from a quadrilateral on the table as indicated by the four clicked points. The bounds of this volume are used to segment the foreground (i.e., the user's hands and the block structure) from the background, without requiring the background to remain static. The depth image is smoothed using a uniform kernel, and the surface normals are computed using the method described in [13].

4.2 Lattice Orientation

The goal of this step of the algorithm is to determine the orientation of the lattice that fits to the building block structure. Intuitively, this amounts to finding the dominant orientation of the surface normals as illustrated in Figure 5(c, g).

We discard the normal vectors that are not parallel to the table surface, such that each remaining normal vector roughly lies on a unit circle in the XZ plane (see Figure 5(d)) and can be represented as a complex number $q_{xz} = (\hat{\mathbf{n}}_x, \hat{\mathbf{n}}_z)$. Normal vectors of points observed on the surfaces of the rectangular building blocks will form four orthogonal clusters, corresponding to the possible surface orientations of the blocks. We find these orthogonal clusters by winding the normal vectors around the unit circle four times (i.e., computing $(q_{xz})^4$), which causes the four clusters to collapse into a single larger cluster as illustrated in Figure 5(e). Then these quantities are averaged, normalized to lie on the unit sphere, and taken to the fourth root to produce a basis q'_{xz} in the XZ plane:

$$q'_{xz} = normalize\left(\left(\frac{1}{N}\sum_{\forall q_{xz}} (q_{xz})^4\right)^{\frac{1}{4}} \right). \tag{2}$$

The rotation matrix $\mathbf{R}$ is simply the matrix representation of the rotation in the XZ plane by q'_{xz}. This matrix can be used to compute the oriented points and surface normals:

$$\mathbf{P}^{oriented} = (\mathbf{R})\mathbf{P}, \qquad \hat{\mathbf{n}}^{oriented} = (\mathbf{R})\hat{\mathbf{n}}. \tag{3}$$

The points can now be labeled according to whether their surface normals align with the X or the Z axes, as illustrated by the red and blue shaded cones in Figure 5(f, g). The radius of the cones used to determine this labeling is a threshold parameter that we have determined experimentally, $\theta_T = \pm 0.3$. We discard any outliers that do not align with either axis (i.e., points that lie outside the shaded cones). Discarding the outliers in this way eliminates many spurious measurements that occur when the user's hands are in the field of view.

4.3 Lattice Translation

We define $(p1_x, p1_z) = \mathbf{P}^{oriented}_X$ and $(p2_x, p2_z) = \mathbf{P}^{oriented}_Z$ to be the points whose normal vectors align with the X and Z axes (i.e., the red and blue points in Figure 5(h)). We solve for each of the lattice translation parameters independently. Since the blocks lie flat on the table surface, the Y component of translation is zero by assumption. In this step, we only need to determine the alignment of the lattice (i.e., modulo the width of a block (w_x)). In other words, we want to find the translation components of the lattice, $t_X \in [0, w_x)$ and $t_Z \in [0, w_z)$, that minimize the distance from each point to the nearest lattice plane, as illustrated in Figure 5(h). By mapping the translation components of each $p1_x$ and $p2_z$ to points on the complex unit circle,

$$\begin{aligned} q_x &= e^{\frac{2\pi i}{w_x} p1_x} \\ q_z &= e^{\frac{2\pi i}{w_z} p2_z}. \end{aligned} \tag{4}$$

We can compute the translation parameters t_X and t_Z that optimally align the lattice to the data in a least squares sense as

$$\begin{aligned} t_X &= arg\left(\frac{1}{N}\sum_{\forall q_x} q_x\right) \cdot \frac{w_x}{2\pi} \\ t_Z &= arg\left(\frac{1}{N}\sum_{\forall q_z} q_z\right) \cdot \frac{w_z}{2\pi}, \end{aligned} \tag{5}$$

where $arg(q)$ is the angular component of a complex number q, in the range $[0, 2\pi)$. The mean values will not generally lie on the unit circle, however the translation parameters are computed only from the arg component. The translation matrix $\mathbf{T}$ is constructed from these components so that

$$\mathbf{P}^{aligned} = (\mathbf{T})(\mathbf{R})\mathbf{P}. \tag{6}$$

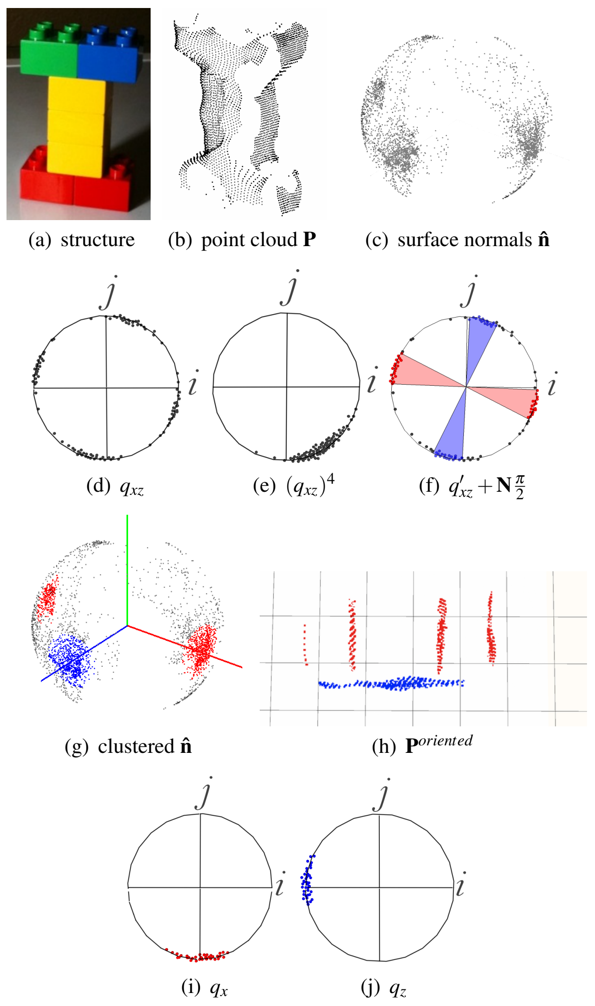

Fig. 5. Visualizations of the key data structures in the Lattice-First algorithm. A building block structure (a) is observed by a depth sensor producing the point cloud measurements $\mathbf{P}$ (b). The surface normals $\hat{\mathbf{n}}$ (c) are projected on the in the XZ plane and represented as complex numbers (d). These values are raised to the power of 4 (causing the clusters to merge (e)) and averaged to find the correct orthogonal clusters of the normals (f,g) (see Section 4.2). The surface points can be labeled as aligning with the X or Z axis (red or blue) or else discarded (black). The oriented point cloud $\mathbf{P}^{oriented}$ (h) is used to find the translation parameters of the aligned lattice by averaging the X and Z coordinates of the points (i,j), modulo the width of the unit block size (See Section 4.3).

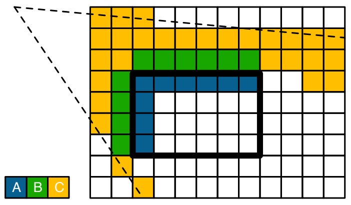

Fig. 6. Illustration of the voxelization process. A solid object (thick black outlines) is observed by a sensor with an indicated viewing frustum. Direct binning (Section 4.4) uses surface points to labels voxels occupied (A) or vacant (B). A simplified variant of space carving (Section 4.5) is used to label additional voxels vacant (C).

The magnitude of the computed means from Equation 5 are used to measure the confidence in the lattice alignment; a magnitude near 1 indicates an unambiguous estimate whereas a magnitude closer to the origin indicates an ambiguous distribution. The voxel grid is only updated when the lattice alignment is confidently estimated (Section 4.7).

4.4 Direct Binning

Once the aligned points $\mathbf{P}^{aligned}$ are computed, a natural voxel grid is established for the blocks. Each observed point corresponds to a surface between two grid cells and can be considered evidence of the *occupancy* of one voxel and the *vacancy* of another. The observed points are binned to the nearest voxel surface and tallied. The tally for each voxel is compared to a threshold $T_V = 30$, which was experimentally determined, to obtain binary values for *occupancy* and *vacancy* of each voxel. These binary estimates are illustrated in Figure 6, where the voxels labeled A are estimated *occupied* and voxels labeled B are estimated *vacant*.

4.5 Space Carving

The direct binning step only produces estimates on or adjacent to the occupied blocks, when in fact many more voxels can be marked *vacant* using information from the depth image. We implement a variant of space carving [18] in which the depth image is sampled once for each voxel grid space. Space carving is typically used to solve a more difficult problem of building a voxel model from color images in which the depth of each pixel is ambiguous. Our adaptation of this technique is simpler: the centerpoint of each voxel is projected onto the depth image and compared to the corresponding depth measurement. If the depth measurement is farther from the camera than the projected centerpoint, then the voxel is marked *vacant*. The voxels marked *vacant* as a result of this step are illustrated in Figure 6 by voxels labeled C.

4.6 Feature-based Alignment

In order to align the current frame with the previous model, we only need to consider translations by integer multiples of $w_x = w_z$, and rotations by a multiple of 90°. This is a discrete rather than a continuous solution space. The voxel model of the current frame represents only a partial view, with some pixels marked *occupied* or *vacant* but many left unmarked or *occluded*. The previous model estimate may be complete if the structure has already been rotated around 360 degrees, or it may contain holes and incomplete information. We perform alignment between a pair of voxel models by minimizing a cost function that rewards matching *occupied* voxels, penalizes disagreement (*occupied* in one model and *vacant* in the other), and ignores voxels that are unmarked or occluded:

$$cost = \sum_v (O_v \cdot V'_v) + (V_v \cdot O'_v) - 0.5(O_v \cdot O'_v) \tag{7}$$

where O_v and V_v are the previous binary *occupancy* and *vacancy* estimates for each voxel v, and O'_v and V'_v are the binary estimates for the current frame.

Instead of evaluating this objective function for each possible translation and rotation, we select a subset of these possible alignments based on sparse feature points computed for both models. The sparse feature points we use are XZ corners (i.e., voxels marked *occupied* that are adjacent to two *vacant* or unmarked neighbors). Each pairing of feature points between the two images, $(f, f') \in \{\mathbf{F} \times \mathbf{F}'\}$, indicates a candidate solution (i.e., a rotation and translation in the XZ plane). These candidate solutions are not unique; we rank the possible feature alignments by the number of instances, and evaluate the objective function at only the $N_F = 40$ highest ranking feature alignments, where N_F is a value experimentally found to work well.

The alignment that minimizes the objective function is used to construct the lattice ambiguity correction matrix:

$$\mathbf{C} = \begin{bmatrix} \mathbf{R}_C & \mathbf{T}_C \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{8}$$

where $\mathbf{R}_c$ is a 3x3 rotation matrix about the Y axis by a multiple of $90°$, and $\mathbf{T}_C$ is a translation in the XZ plane (i.e., $\mathbf{T}_C = (t_x, 0, t_z)$). This correction matrix is used to find the final *model coordinates*

$$\mathbf{P}^{model} = (\mathbf{C})(\mathbf{T})(\mathbf{R})\mathbf{P}. \tag{9}$$

4.7 Backprojection and Model Update

The properly aligned voxel grids may still disagree between the previous estimate and the current frame. In the case where a voxel in the previous estimate is unmarked or occluded, but in the current frame it is *vacant* or *occupied*, then the new frame provides new information and the model is easily updated. However in the cases where the previous estimate and the current estimate disagree, it may be due to the user's hands occupying a previously vacant space or caused by an added or removed piece. Our method for distinguishing between these cases is to render with OpenGL the union of the *occupied* voxels in both the previous and the current estimate, producing a backprojected virtual depth image that can be compared with the observed depth image. This approach is similar to one used in [41]. The color buffer is used to store the index of each voxel so visible pixels are easily associated to corresponding voxels. Each virtual depth value is compared to the observed depth value, and the absolute difference is thresholded. The ratio of the number of pixels matching the rendered image for each voxel, $N_v^{matched}$ to the expected number of visible pixels for each voxel, $N_v^{expected}$ is used in the following decision:

$$accept(v) = \left(\frac{N_v^{matched}}{N_v^{expected}} > \alpha_V \right) \wedge \left(N_v^{expected} > T_V \right) \tag{10}$$

where T_V is the same threshold used in Section 4.4 to indicate a significant number of pixel observations, and $\alpha_V = 0.9$ is a parameter we have experimentally found to work well.

5 Prototype Implementation

Our system uses a Microsoft Kinect depth camera arranged to point at a table surface, as shown in Figure 1. The camera provides a 640x480 depth image, with 10 bits of resolution. Although our system uses only depth information for tracking and acquisition, the Kinect additionally provides RGB data aligned to the depth image which can be used to determine the color of each voxel. Our treatment of color is simple: using the mapping from pixels to voxels found in Section 4.7, we find the average RGB values for each voxel and round to the nearest primary color in HSV space.

We implemented our algorithm using a number of software components. The overall system is written in Python, along with several routines written in C or Cython, and several others implemented on the GPU using OpenCL/PyOpenCL. The software runs on a desktop computer with an Intel i7-700 processor with an nVidia 590GTX GPU. Realtime visual feedback is provided to the user by means of an OpenGL display in which the vitual block structure is rendered. We also display the block structures as an interactive HTML5 canvas, using the Three.js library.

The proposed algorithm is essentially GPU-friendly because it involves several computations over all pixels in a rectangular image and a small number of reductions to compute the means of the circular quantities. The surface normal computation, lattice orientation, lattice translation, and direct binning stages were implemented using OpenCL (see Sections 4.1 through 4.4). The full interactive system runs at 25 frames per second, although the algorithm itself (excluding image capture and the interactive display) runs at 50 frames per second.

The source code to our prototype implementation is provided as supplemental material. Since our code relies on a number of libraries, notably OpenCL, we have packaged up our system as an "Experiment in a Box," a virtual machine image with our code and necessary libraries configured and installed. This virtual machine image can be downloaded on the Interactive Systems and User Experience website: `http://eecs.ucf.edu/isuelab/`.

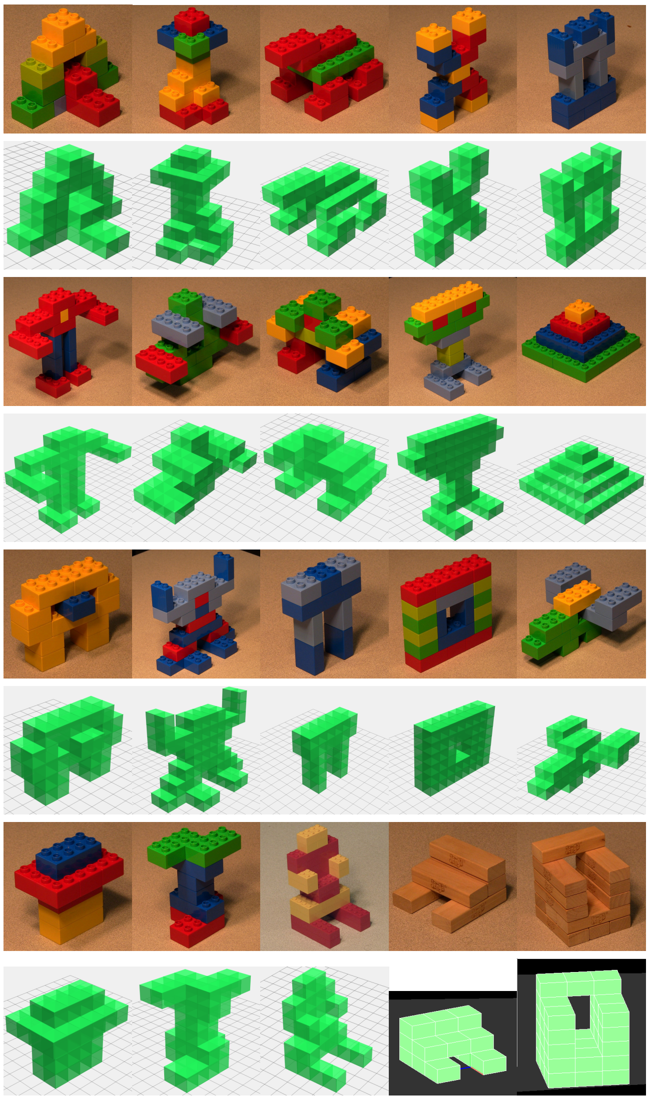
Fig. 7. Examples of building block models correctly acquired by our system.

6 Results

6.1 Examples of Acquired Models

Twenty examples of 3D building block structures acquired correctly using our interactive system are shown in Figure 7. For each of these structures, the user placed the completed block structure in front of the sensor, initializing the virtual model with a partial view, and then rotated the object on the table until the virtual model was fully reconstructed. In some cases it was necessary to remove and later replace a piece in order to reveal hidden structure, such as underneath the overhang in Figure 7 (bottom row, leftmost column). The visual feedback was used to know where there were errors that needed to be corrected or holes to be filled in, however the only user input was through manipulating the physical structure.

These structures primarily use Duplo blocks, however we show some examples of structures made of other types of blocks that satisfy our assumptions: Jenga blocks, which have a unit dimension of $(w_x, w_y, w_z) = (20mm, 15mm, 20mm)$ (Figure 7, bottom row, first and second from the right); and Lego blocks, which share the same unit size as Duplo blocks when arranged in pairs (Figure 7, bottom row, center). Building blocks with smaller dimensions are not reliably tracked because of the limited resolution of our sensor.

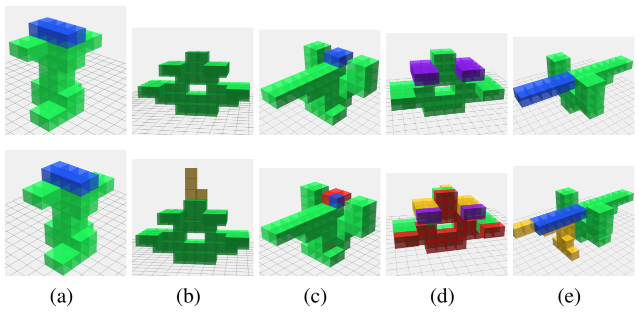

Fig. 8. Examples of ground truth and comparison with estimated results in our experiment. Top row: ground truth, where blue indicates a block that was added and purple indicates a block that was removed. Bottom row: results of our algorithm, where yellow and red indicate errors (false positives and false negatives); green blue and purple indicate correct estimates.

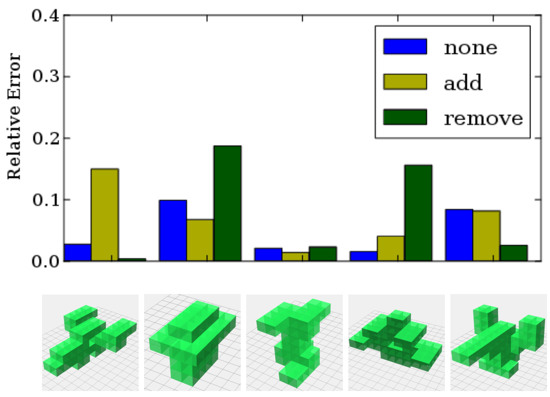

Fig. 9. Results of an experiment in the absence of visual feedback. Relative error is computed as number of incorrect voxels divided by the number of voxels in ground truth, averaged across the four participants used as a validation set.

6.2 Experiment in the Absence of Visual Feedback

In the previous section, we showed that a variety of structures can be acquired using the interactive system. We also wanted to evaluate how effective the system is in a more challenging scenario, in the absence of visual feedback to the user. For interactive assembly tasks, users will need to consume visual information in the form of instructions and ideally would not require visual feedback or user intervention in the acquisition process itself.

We asked five participants to manipulate a building block structure while video was recorded; no visual feedback was provided, and our algorithm was applied to the recorded data afterward. We provided five block structures and asked the participants to perform three trials for each shape, fifteen seconds per trial. In the first trial, the participant would rotate the complete structure without modifying it; in the second, the participant would add a single piece to the original structure; in the third, a single piece was removed.

Participants were shown a short video clip demonstrating each action and reminding them to rotate the object at least 360°, but otherwise were not directed to move the structure in a particular manner or speed. One of the participants' data was used as a *tuning set* during the development of our algorithm, in particular to determine acceptable values for the threshold parameters mentioned in Section 4. The algorithm was applied to the remaining four data sets which were used for validation. Ground truth files were created by hand, and the algorithm was initialized with the correct ground truth so that the experiment reflects the use of the system after the virtual model has been initially acquired for the original structure. The ground truth files contain one

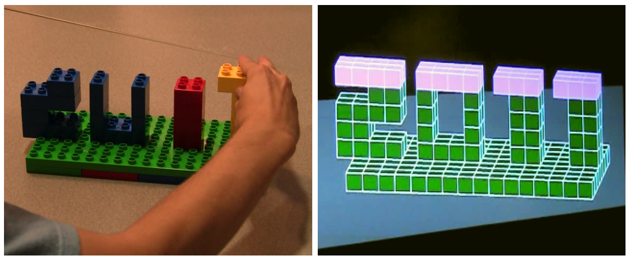

Fig. 10. Our proof-of-concept application for collaborative guided assembly. The user adds blocks layer by layer (left) while the system highlights the next layer of blocks to add on the rendered virtual model (right).

of three labels for each voxel: *occupied*, initially *vacant* but with a piece added, initially *occupied* but with a piece removed. The final frame of each trial is scored against ground truth. Several examples are shown in Figure 8 (top row) with voxel labels indicated by a color code.

The results of the experiment are summarized in Figure 9, where relative error is calculated as the number of erroneous voxels (indicated by red or yellow) divided by the number of *occupied* voxels in the ground truth, and the relative errors are averaged across the four *validation* participants. Although our system correctly estimates a majority of the voxels under these conditions, several errors do occur. Some examples are illustrated in Figure 8 using a color code where errors are indicated by red or yellow (false negatives and false positives) and correct estimates are indicated by green, blue (added piece), or purple (removed piece). In Figure 8(b), the user's hand is falsely considered a voxel, while in Figure 8(c) only one corner of an added piece is correctly detected. In Figure 8(d), a tracking misalignment caused half of the voxels to be erroneously discarded; this may be due to the fact that the participant spun the structure around very quickly, like a top. Figure 8(e) shows a difficult case where an added piece makes the structure become symmetrical and a tracking error caused misalignment by 180°. These results indicate that more robustness is needed in the absence of visual feedback for an ideal system and is an area of future work.

6.3 Collaborative Guided Assembly

We implemented a proof-of-concept application that interactively assists the user in assembling a physical replica of a block structure previously acquired by our system. The user builds the model from the table up, layer by layer, while the system highlights the next layer to build on the virtual model (see Figure 10). Our realtime acquisition system is used to determine when the user has completed building a layer and is ready to move to the next layer. With this system, one user can create a physical structure while another user at a remote location is guided through constructing a replica.

6.4 Hole-in-the-Wall Game

We also implemented a proof-of-concept game in which our model acquisition system is used as the input interface (see Figure 11). In our game, the player must build a block structure that matches a silhouette carved out of a wall that advances towards the virtual model. The structure must be correctly assembled by the time the wall reaches it so that it passes over without colliding. This gameplay is inspired by the television gameshow Hole-in-the-Wall.

7 Discussion

7.1 Comparison of Lattice-First with Other Algorithms

Unlike other realtime 3D model acquisition systems, the Lattice-First method supports robust model acquisition while *simultaneously* allowing natural user interaction with the object. In other words, Lattice-First lets users add and remove pieces of a 3D model with their bare hands as it is continuously acquired. KinectFusion [15, 26], on the

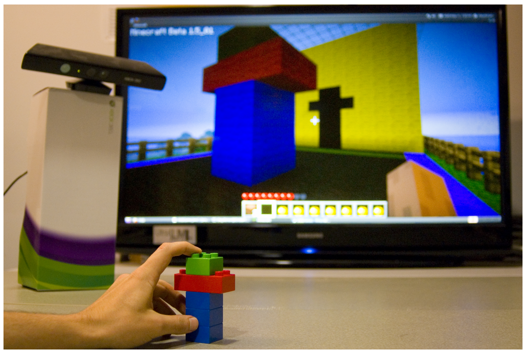
Fig. 11. An application of our system as a construction gameplay interface. A silhouette is shown on the screen and the player must build a matching physical model within a time limit. This application was implemented as a modification to Minecraft.

other hand, supports realtime 3D model acquisition of a static scene, *followed by* user interaction with parts of the previously-scanned model. Therefore KinectFusion does not support the user-interaction workflow that the Lattice-First method affords. Instead, KinectFusion has a *moving average* parameter that can be adjusted to control whether the model reconstruction converges after some time, after which any motion in the scene is discarded as outliers, or whether the model is continuously updated with new data. In the former case, modifications to the structure after convergence would be discarded, while in the latter, the user's hands would be erroneously incorporated into the model along with the actual modifications. Lattice-First addresses this problem by using *a priori* knowledge of the lattice-like structure of the model to discard pixels from the user's hands (see Section 4.2) while tracking and acquiring the target object. Additionally, the fact that KinectFusion assumes a stationary scene and a moving camera during acquisition means that more datapoints are available to support registration than in our scenario. To support our user interaction workflow, our algorithm must track the model using far fewer data points as well as coping with outliers during acquisition, as illustrated in Figure 12.

The registration component of a model acquisition system typically involves a linear least squares fit between the incoming range data and the continuously updated model (e.g., the commonly used FastICP algorithm) [15, 32, 33, 41]. Realtime model acquisition systems are typically susceptible to 'drift', a positive feedback loop that occurs when slight registration errors cause errors to accumulate in the acquired model, and vice versa. Our method computes a partial estimate of the orientation and translation parameters (up to the symmetry of a lattice) by fitting a lattice to each incoming frame directly, without comparison to the previous model estimate (see Sections 4.1 through 4.5). Our method is thus able to tolerate the small tracking errors (caused by the remaining outliers from the user's hands) without accumulating error. This is particularly important for supporting the addition and removal of pieces, as a dynamic model is more susceptible to positive feedback. Note that while our algorithm is robust against the accumulation of small tracking errors, larger errors in tracking may still result in accumulated errors in the estimated model (see Figure 8).

An additional difficulty of our user interaction scenario that Lattice-First addresses is that the block structures are degenerate shapes when viewed from certain angles with respect to our registration algorithm. When a block structure is rotated to face the sensor so that only one of the XY or ZY planes is visible, then the registration tracking is unsuccessful. It was demonstrated in Rusinkiewicz et al. [32] that FastICP as well is also unable to correctly converge in the case of a single flat planar surface. Thus, during user interaction, tracking will occasionally be interrupted and must be restarted in some way. In Rusinkiewicz et al. [32], the user is guided by online feedback to align the physical object with a virtual "anchor frame," explicitly requiring the user's active effort in order to regain tracking. Our system performs a global alignment with each new frame, which allows tracking to resume automatically without the user's attention.

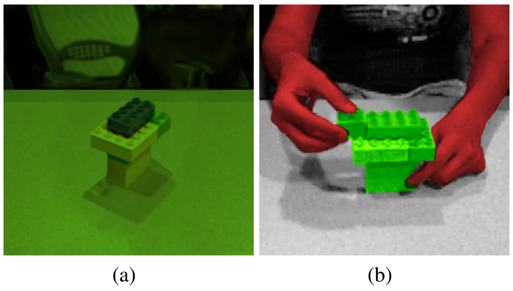
Fig. 12. Illustration of inliers and segmentation requirements for the 3D model acquisition problem under the assumptions of (a) KinectFusion [15] and (b) our system (Lattice-First). Green-colored pixels are inliers that support an accurate tracking estimate, grey pixels must be discarded using a *volume of interest* (see Section 4.1), and red pixels must be distinguished from the target model and discarded. In (a), the camera moves but the model is stationary, and therefore the entire scene, including table and the chair in the background, supports the tracking estimate. Our method, however, is able to robustly track the object even when it moves relative to the background and the user's hands are in view (b).

7.2 Limitations of Lattice-First

Our algorithm is ultimately constructed to fully exploit the restricted domain of building blocks. This decision represents a tradeoff between generality in terms of the 3D models we can acquire and the ability to have the user interact with the model as it is being constructed and acquired. However, even with this limitation, our approach has several potential uses (see Section 7.3).

The minimum size of the blocks (i.e., the voxel resolution) we can accurately acquire is dependent on the resolution and precision of the depth camera. Using the Kinect sensor, we find we are able to reliably track blocks that are $(16mm)^3$. However, we are unable to acquire complex models built using smaller blocks because our algorithm relies on a local-neighborhood normal vector computation. The low precision of the Kinect sensor requires us to filter the image, limiting the smallest size of blocks we can detect. Our algorithm has no inherent limitations regarding the maximum resolution, so using blocks with larger dimensions would be possible. It is not necessary for the entire block model to fit in the image at all times. Since our model representation is a voxel grid, memory requirements grow linearly with the volume we acquire. For example, using $(16mm)^3$ blocks, we could acquire a model, 8 meters on each side, requiring 512^3 voxels and a memory footprint of 512 megabytes.

The placement of the sensor relative to the table surface has an impact on the quality of model acquisition. If the view from the camera is parallel to the table, then the tops of blocks will not be visible, and therefore some holes will not be filled in. Of course, if the camera were mounted vertically looking down, then the sides of the blocks would not be visible. In our experience, placing the camera about a half meter off the table and facing down at 45 degrees works well.

7.3 Potential Uses

Our system has several potential uses in mixed or augmented reality. Both our proof-of-concept applications rely on the ability to recognize a 'correct' or 'incorrect' structure to give specific feedback. This feedback is reliable because of the relative unambiguity of the building blocks, for example compared to modeling clay. Once assembled, the tracked structure can also function as a positional input device. Our guided assembly proof-of-concept demonstrates how our system can apply to collaborative mixed-reality, where physical objects assembled in one environment can be transmitted and replicated in another environment.

Research has suggested that users perform some assembly tasks more effectively when guided by an augmented reality display [38]. Several other projects have involved augmented reality to assist in physical tasks, including assembly and repairs [10, 28, 30], or as a projected interactive tabletop [16, 29, 45]. These systems lack robust realtime 3D model acquisition, and therefore do not support interactive construction of new models. We believe that Lattice-First has the ability to enhance these systems.

We believe our current work is also applicable to a number of projects that use building block structures as tangible interfaces. Building block models have been used as interactive aids in STEM education, such as by modeling chemical structures or by simulating protein synthesis [5, 39]. Even though building block structures can only coarsely approximate arbitrary objects, their familiarity, tangibility, and ease-of-use makes them appealing and versatile modeling tools. Building block models also naturally support collaboration between multiple users. They have been used in this context as therapy to improve the social competency of autistic children [20], and as a roleplay-based simulation to train first year medical students in patient-interview techniques [12]. By using Lattice-First to acquire these models, we could enhance these systems with interactive user feedback (e.g., intelligent tutoring).

8 Future Work

There are several ways in which we can improve on the Lattice-First algorithm. For example, we would like to extend this algorithm to work in the unconstrained case with 6 degrees of freedom. In such a system, the user would be able to pick up the structure from the table while it is continuously tracked and updated, holding the structure in one hand while adding pieces with the other. It may be possible to relax other constraints to support a wider variety of structures. It would also be desirable to track and acquire more than one model at a time in a scene.

We intend to use this block structure acquisition system as we work towards the goals described in the introduction. Specifically, we want to use an augmented reality display to provide visual feedback to the user, superimposed directly on top of the physical blocks. This should remove a cognitive step that is typically required when a user assembles a physical object while referencing printed instructions. Registered projected imagery, as in [29], could apply color and texture that appears fixed to the block structure as it moves.

The results of our preliminary experiment suggest that users must identify errors in the rendered virtual model in order to correct for acquisition errors. We would therefore like to improve our system in order to achieve better results without burdening the user to correct for errors.

9 Conclusion

We have presented a prototype system for acquiring and tracking 3D physical models made up of building blocks. Users can interactively construct the models using their hands while the system acquires additions or deletions to the model. The system makes use of the Lattice-First algorithm, a novel approach that uses the orthogonal properties of building blocks to estimate alignment parameters directly for each frame. The algorithm makes use of a depth sensing camera and applies a series of transformations to raw depth images to find the physical model in an axis-aligned 3D grid. We have also discussed a proof-of-concept application where users are provided instructions to build a physical model that another user has designed, and a game where users must construct a model that fits through a silhouette carving of a wall. We believe our prototype lays a solid foundation for ubiquitous augmented reality and natural interaction in guided assembly of 3D physical models.

Acknowledgements

This work is supported in part by NSF CAREER award IIS-0845921 and NSF awards IIS-0856045 and CCF-1012056. We would also like to thank the members of the ISUE lab for their support and the anonymous reviewers for their useful comments and feedback.

References

[1] D. Anderson, J. Frankel, J. Marks, A. Agarwala, P. Beardsley, J. Hodgins, D. Leigh, K. Ryall, E. Sullivan, and J. Yedidia. Tangible interaction+ graphical interpretation: a new approach to 3D modeling. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 393–402, 2000.

[2] P. Baudisch, T. Becker, and F. Rudeck. Lumino: tangible blocks for tabletop computers based on glass fiber bundles. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1165–1174, New York, NY, USA, 2010. ACM.

[3] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:239–256, February 1992.

[4] M. Bray, E. Koller-Meier, and L. J. V. Gool. Smart particle filtering for high-dimensional tracking. *Computer Vision and Image Understanding*, 106(1):116–129, 2007.

[5] D. J. Campbell, J. D. Miller, S. J. Bannon, and L. M. Obermaier. An Exploration of the Nanoworld with LEGO Bricks. *Journal of Chemical Education*, pages 602–606, 2011.

[6] R. J. Campbell and P. J. Flynn. A survey of free-form object representation and recognition techniques. *Comput. Vis. Image Underst.*, 81(2):166–210, February 2001.

[7] G. K. Cheung, S. Baker, and T. Kanade. Visual hull alignment and refinement across time: A 3d reconstruction algorithm combining shape-from-silhouette with stereo. *Computer Vision and Pattern Recognition*, 2:375, 2003.

[8] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques SIGGRAPH 96*, pages(Annual Conference Series):303–312, 1996.

[9] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 145–156, New York, NY, USA, 2000. ACM.

[10] S. Feiner, B. Macintyre, and D. Seligmann. Knowledge-based augmented reality. *Commun. ACM*, 36:53–62, July 1993.

[11] S. Fuchs and G. Hirzinger. Extrinsic and depth calibration of tof-cameras. In *Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2008.

[12] S. R. Harding and M. F. D'Eon. Using a LEGO-based communications simulation to introduce medical students to patient-centered interviewing. *Teaching and Learning in Medicine*, 13(2):130–135, 2001.

[13] B. K. P. Horn and J. G. Harris. Rigid body motion from range image sequences. *CVGIP: Image Underst.*, 53:1–13, January 1991.

[14] H. Ichida, Y. Itoh, Y. Kitamura, and F. 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.

[15] S. Izadi, A. Davison, A. Fitzgibbon, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, and D. Freeman. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, pages 559–568, New York, New York, USA, Oct. 2011. ACM Press.

[16] R. Jota and H. Benko. Constructing virtual 3d models with physical building blocks. In *CHI 2011 Extended Abstracts*, pages 2173–2178. ACM, May 2011.

[17] S. B. Kang and K. Ikeuchi. The complex egi: A new representation for 3-d pose determination. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15:707–721, July 1993.

[18] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *Int. J. Comput. Vision*, 38:199–218, July 2000.

[19] J. Lee, Y. Kakehi, and T. Naemura. Bloxels: glowing blocks as volumetric pixels. In *ACM SIGGRAPH 2009 Emerging Technologies*, SIGGRAPH '09, pages 5:1–5:1, New York, NY, USA, 2009. ACM.

[20] D. B. LeGoff. Use of LEGO as a therapeutic medium for improving social competence. Technical Report 5, Bancroft Neurosciences Institute, Haddonfield, NJ 08034, USA. dlegoff@bnh.org, 2004.

[21] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 131–144, New York, NY,

USA, 2000. ACM.

[22] X.-Y. Li, C.-H. Shen, S.-S. Huang, T. Ju, and S.-M. Hu. Popup: automatic paper architectures from 3d models. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 111:1–111:9, New York, NY, USA, 2010. ACM.

[23] M. R. Marner and B. H. Thomas. Augmented foam sculpting for capturing 3d models. *IEEE Symposium on 3D User Interfaces*, pages 63–70, April 2010.

[24] Y. Mori and T. Igarashi. Plushie: an interactive design system for plush toys. *ACM Trans. Graph.*, 26:45:1–45:8, July 2007.

[25] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. *IEEE International Conference on Computer Vision*, 1, 2011.

[26] R. A. Newcombe, D. Molyneaux, D. Kim, A. J. Davison, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. *ISMAR*, pages 127–136, 2011.

[27] H. Newton-Dunn, H. Nakano, and J. Gibson. Block jam: a tangible interface for interactive music. In *Proceedings of the 2003 conf. on New interfaces for musical expression*, NIME '03, pages 170–177, Singapore, Singapore, 2003. National University of Singapore.

[28] R. Raskar, P. Beardsley, P. Dietz, and J. van Baar. Photosensing wireless tags for geometric procedures. *Commun. ACM*, 48:46–51, September 2005.

[29] R. Raskar, G. Welch, and W. chao Chen. Table-top spatially-augmented reality: Bringing physical models to life with projected imagery. In *In: Proceedings of the 2nd IEEE and ACM international workshop on augmented reality (IWAR99)*, pages 64–73. IEEE, 1999.

[30] S. Rosenthal, S. K. Kane, J. O. Wobbrock, and D. Avrahami. Augmenting on-screen instructions with micro-projected guides: when it works, and when it fails. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, Ubicomp '10, pages 203–212, New York, NY, USA, 2010. ACM.

[31] F. Rottensteiner and C. Briese. A new method for building extraction in urban areas from high-resolution lidar data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):295301, 2001.

[32] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3d model acquisition. *ACM Trans. Graph.*, 21:438–446, July 2002.

[33] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3D Digital Imaging and Modeling*, pages 145–152. IEEE, 2001.

[34] M. Schlattmann, F. Kahlesz, R. Sarlette, and R. Klein. Markerless 4 gestures 6 dof real-time visual tracking of the human hand with automatic initialization. *Comput. Graph. Forum*, 26(3):467–476, 2007.

[35] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[36] E. Sharlin, Y. Itoh, B. Watson, Y. Kitamura, S. Sutphen, and L. Liu. Cognitive cubes: a tangible user interface for cognitive assessment. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, CHI '02, pages 347–354, New York, NY, USA, 2002. ACM.

[37] J. Slaney and S. Thiébaux. Blocks world revisited. *Artif. Intell.*, 125:119–153, January 2001.

[38] A. Tang, C. Owen, F. Biocca, and W. Mou. Comparative effectiveness of augmented reality in object assembly. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 73–80, New York, NY, USA, 2003. ACM.

[39] M. A. Templin and M. K. Fetters. A Working Model of Protein Synthesis Using LEGO. *The American Biology Teacher*, 64(9):673–678, 2002.

[40] R. Y. Wang and J. Popović. Real-time hand-tracking with a color glove. *ACM Trans. Graph.*, 28:63:1–63:8, July 2009.

[41] T. Weise, T. Wismer, B. Leibe, and L. J. V. Gool. Online loop closure for real-time interactive 3d scanning. *Computer Vision and Image Understanding*, 115(5):635–648, 2011.

[42] G. Welch and E. Foxlin. Motion Tracking: No Silver Bullet, but a Respectable Arsenal. *IEEE Computer Graphics and Applications*, 22(6):24–38, 2002.

[43] M. P. Weller, E. Y.-L. Do, and M. D. Gross. Posey: instrumenting a poseable hub and strut construction toy. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 39–46, New York, NY, USA, 2008. ACM.

[44] S. Wesugi and Y. Miwa. Brick-building interface support for cocreative communication. *Int. J. Hum. Comput. Interaction*, 20(1):35–56, 2006.

[45] R. Ziola, S. Grampurohit, N. Landes, J. Fogarty, and B. Harrison. Oasis: Examining a framework for interacting with general-purpose object recognition. In *Intel Labs Seattle Technical Report*. Intel, 2010.