Multimed Tools Appl (2014) 73:2009–2032
DOI 10.1007/s11042-013-1690-7

SnapBlocks: a snapping interface for assembling toy blocks with XBOX Kinect

Juncong Lin · Qian Sun · Guilin Li · Ying He

Published online: 7 September 2013

Abstract Toy blocks can help the children develop various skills, such as spatial, mathematical, creative problem solving etc. In this paper, we developed a computer aided system for child to play blocks with a computer in a natural and intuitive way using the Kinect. We design a set of intuitive body gestures that allow the user to naturally control and navigate 3D toy blocks in a virtual environment. To conquer the imprecise interaction with Kinect, we propose a snapping interface, which automatically computes the optimal location and orientation of the to-be-assembled block. This interface can significantly reduce the user's burden for fine tuning the blocks at the desired locations, which is often tedious and time consuming. As a result, the user can fully immerse him/herself in the game and construct a complicated structure easily. The experimental results and positive feedback from users demonstrate the efficacy of our approach to virtual assembly of building blocks.

Keywords Toy Blocks · Kinect · Snapping interface

J. Lin (✉) · G. Li
Software School, Xiamen University, Xiamen, People's Republic of China
e-mail: JuncongLin@gmail.com

G. Li
e-mail: glli@xmu.edu.cn

Q. Sun · Y. He
School of Computer Engineering, Nanyang Technological University, Nanyang, Singapore

Q. Sun
e-mail: SUNQ0004@ntu.edu.sg

Y. He
e-mail: yhe@ntu.edu.sg

1 Introduction

Toy blocks provide children with the opportunity to exercise a variety of skills (e.g., math, space, problem solving, etc) that will help them later in life. Blocks also motivate creative thinking and encourage social, emotional, and cognitive skills. Block building potentially provides one of the most valuable learning experiences available for young children [16]. Developing a computer aided system for block building is desirable for recreational and educational applications. With such a system, users (especially kids) are given a lot of freedom so they can enjoy the process of imagining, designing and constructing with blocks freely. They can easily try with various block sets without purchase. They can even try those not available in the market. However, block building in computer with current dominant direct manipulation interaction style could be awkward. Users are confined to their desktop computer, sitting at a chair with mouse and keyboard on the desk, viewing a display in front of them. Such an interaction style may be powerful for some applications (text editing, spreadsheets), however, it is not suitable for recreational applications that involves lots of body movements. Besides, as the direct manipulation interfaces become more complex the metaphors can begin to confuse the user [15]. These issues make the interaction style even more difficult for young children who are the dominant participants in block building. It is difficult and also not appropriate to ask child to sit in front of the computer for long time to play the game. A more natural and more intuitive interaction style is highly desired for entertainments applications targeting on children such as block building. The release of Microsoft Kinect provides a new way to control and interact virtual objects with body gestures. Such a hands-free control has great potential in block building. However, building a Kinect application for block building presents design challenges due to several issues: first, there is no reference object in the real world for the users to feel the relative position of virtual objects in the virtual world; second, the captured 3D videos are imprecise in nature, which makes the reconstructed skeleton unstable and noisy; third, the designed gestures should be robust, efficient, memorable, and comfortable to perform.

In this paper, we present *SnapBlocks*, a Kinect framework of virtual assembly of building blocks, which allows users to interactively and efficiently construct a complicated 3D shape from simple blocks in a virtual environment. Within our framework, the user first picks a building block which is going to be assembled into the virtual structure. Then he controls the avatar who carries the to-be-assembled building block through a set of intuitive body gestures. When the avatar approaches the virtual structure, the user can control avatar to place the current block at the desired location. To conquer the aforementioned imprecision interaction issue, we propose an effective snapping interface allowing users to easily place the building blocks at the desired location. Our snapping interface automatically computes the optimal location and orientation of the current to-be-assembled block by considering various physical and geometric criteria, including distance, alignment, contacting surface, balance and symmetry. We formulate these criteria into a non-linear optimization problem and propose an efficient technique to compute it at interactive speed. The experimental results and positive feedback from users show that our approach is effective for virtual assembly of building blocks. Figure 1 shows an example of constructed 3D building by using our framework.

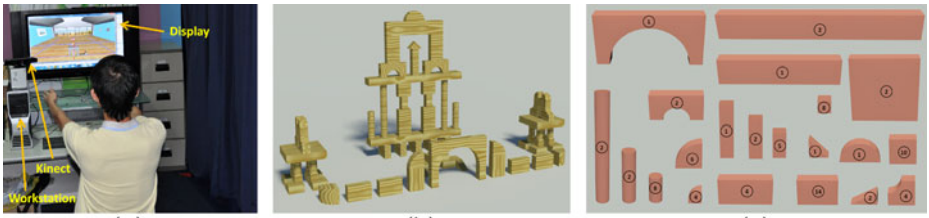

Fig. 1 Assembling building blocks with Kinect. **a** The user is playing with our system. **b** A virtual castle constructed by our system. **c** The set of building blocks for the castle. The value on each block indicates the number of blocks used in this virtual assembly

The contributions of this paper are two-fold:

- First, we present a Kinect framework for virtual assembly of building blocks. We design a set of intuitive body gestures for scene navigation, object selection and manipulation. The designed gestures are memorable, efficient and comfortable to perform.
- Second, we propose an efficient algorithm to automatically compute the optimal location and orientation of the current block, so that the user can easily assemble the block in the virtual environment without the tedious adjustment. Our snapping interface is effective and easy to use.

1.1 Related work

Kinect application The release of Microsoft Kinect makes depth cameras widely available, which significantly stimulates the applications of depth cameras especially for 3D interaction. For example, Wilson explored the application of depth-sensing cameras to detect touch on a tabletop [24]. Fisher and Hartmann developed a mobile window into a virtual world using depth camera [1]. Kinect have also been used as 3D scanners to scan rigid objects such as indoor environments [3, 4, 14, 17, 18] or non-rigid objects such as human faces [22] or bodies [20, 23]. We use Kinect as a full body interaction tool for 3D navigation and manipulation tasks.

3D navigation and manipulation Scene navigation and object manipulation are basic operations in 3D applications. Manipulating virtual objects with mouse controls usually takes more time than direct manipulation of real objects [21]. Haptic devices are highly desirable in navigation and manipulation applications [19] as their provide high degree-of-freedom and sense of touch with force feedback. The emergency and popularity of multi-touch devices provide a flexible and natural way for 3d navigation and object manipulation [5]. Our *SnapBlocks* provides a novel interaction style for navigation in the virtual environment and for manipulation of 3d object with various body gestures without touching or holding any physical object.

Recreational graphics Recently, recreational application(such as papercarft [10], various puzzles [26], plush toys [13], paper architectural models [6], shadow art [11], etc.) has been a popular topic in computer graphics community. We introduce a

new recreational application, *SnapBlocks*, a system for playing toy blocks in an intuitive and natural manner with Kinect. Different from previous applications, *SnapBlocks* could also be used for educational purpose which is actually what our system targets on.

The remaining of the paper is organized as follows: Section 2 presents the system overview, followed by the designed body gestures in Section 3. Next, we document the algorithmic details of our snapping interface in Section 4. Section 5 describes the user study and shows the experimental results. Finally, Section 6 draws the conclusion.

2 System overview

Our system consists of a large-screen display, a Kinect positioned next to the display, and a desktop PC computer as shown in Fig. 2a. The user stands in front of the screen and Kinect, and is allowed to move within the visible area of Kinect. Our system has three modes, i.e., selection, navigation and assembly. In the selection mode (see Fig. 2b), the user can browse all the building blocks from the database (see the window on the left) and then select one block which is going to be assembled in the virtual environment. The window on the right shows a close-up view of the current block. The selection process is done when the user performs a pushing gesture, then our system automatically switches to the navigation mode. In this mode (see Fig. 2c), the user controls the avatar who carries the block towards the desired position. Once the avatar is close to the target, the user performs a gesture to activate the assembly mode, in which the user can manipulate the virtual object and place the block into the virtual structure. To reduce the user's burden in the object manipulation (sometimes, it may be very tedious), our snapping interface automatically computes the optimal location and orientation of the block. With a simple releasing gesture, the user can put the block in the desired location.

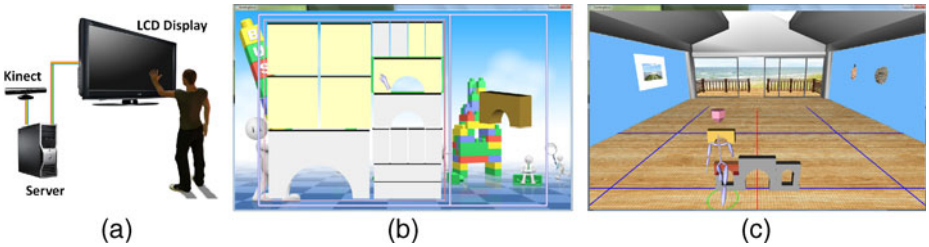

Fig. 2 The proposed virtual assembly framework. **a** System configuration. **b** Selection mode. The block set window is on the *left*. The user can browse all the blocks by freely moving his hand, which controls the cursor in the block set window. The current block is highlighted in a *green box* and the window on the *right* shows its 3D view. The already-assembled blocks are drawn in *gray* and the not-yet-assembled blocks are drawn in *yellow*. **c** Navigation and assembly modes. In the navigation mode (*when the pink cursor is on top of the avatar*), the user controls the avatar who carries the current to-be-assembled block towards the 3D structure in the virtual environment. In the assembly mode (*when the pink cursor is on the current block*), the user can assemble the current block into the structure and fine-tune the location and orientation of the current block. The *green circle* around the avatar indicates the region of interest. The reference grid, lines and symmetry axis are marked on the ground to help user better align the blocks

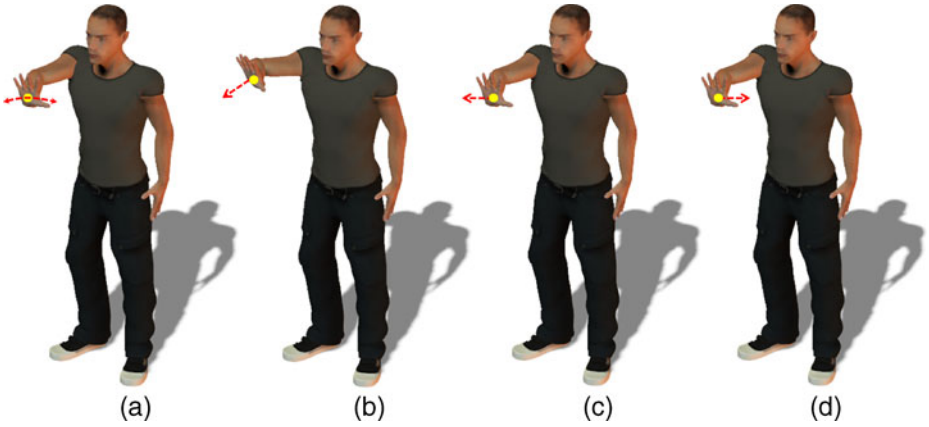

Fig. 3 Selection gestures. **a** Activating the selection mode. **b** Selecting a block. **c–d** Viewing more blocks on other pages

3 Body gestures

We design a set of body gestures for 3D scene navigation and virtual object control. Our body gestures can be classified into three categories: selection, navigation, and assembly. One of the most important principles in gesture design is to make all gestures clearly distinguish from each other, so as to avoid the misinterpretation or ambiguity. Another factor considered in our current system is that the user always faces the screen, so that he can see the virtual scene on the screen.[1] Last but not the least, the designed gestures should be intuitive, and comfortable to perform. The navigation gestures are inspired from road traffic control gesture, while the selection gestures and the assembly gestures mimic the mouse operations and real operating gestures (such as how people hold object with two hands).

Selection gestures are used in the selection mode allowing the user to browse all the available building blocks and select one of them. To activate a selection session, the user quickly waves his hand at least two times (see Fig. 3a). In the selection mode, the user can freely move his hand to control the cursor in the browsing window. To select a block, the user simply pushes his hand forward (see Fig. 3b). Furthermore, the user can also swipe to left or right to view more blocks on other pages (see Fig. 3c, d).

Navigation gestures are used to control the avatar in the virtual environment. In our system, the avatar can walk along the directions of coordinate axes. The walking speed of the avatar is specified by the user before starting the system. As shown in Fig. 4 ,we define five gestures to control the avatar's movement (go straight, turn left, turn right, stop and finish).

Assembly gestures are used to place the block in the virtual structure and further adjust its location/orentation. To do that, the user first performs a holding gesture to

[1]This requirement would not be necessary if we configurate our system in an immersive environment.

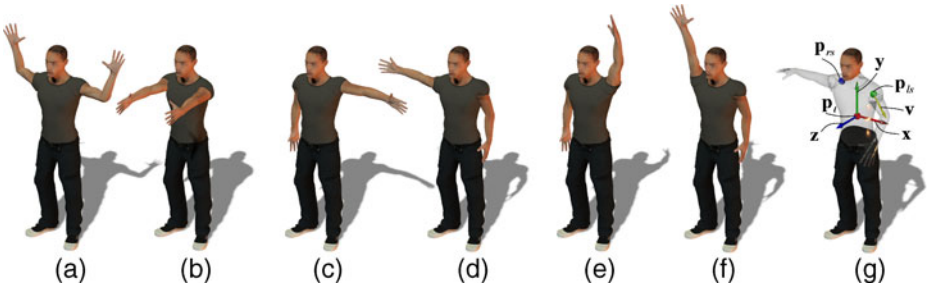

Fig. 4 Navigation gestures. **a** Activating the navigation mode; **b** Going straight; **c** Turning left; **d** Turning right; **e** Stop; **f** Finish the navigation mode; **g** Coordinate system

activate assembly mode (see Fig. 5a). Then the user can manipulate the block with the Rotation and Translation gestures (see Fig. 5b–e). To release the block, the user moves his hands in opposite directions (see Fig. 5f).

We can then compare the current gesture performed by the user with the three body gesture sets mentioned above to determine the instruction given by the user. To be specifically, we check the moving directions of body joints or the directions of the body limbs. However, simply using the default coordinate system defined in the camera space is not appropriate due to the instability of the depth camera and the small variation of the user's posture (See Section 5 for more details). We thus parameterize the gesture in an intrinsic way. We firstly build a coordinate system from the detected skeleton. As shown in Fig. 4f, let $\mathbf{p}_{ls}$ and $\mathbf{p}_{rs}$ be the locations of left and right shoulder respectively. $\mathbf{p}_t$ is the torso joint position. The coordinate system is defined as follows:

$$\begin{cases} \mathbf{z} = \frac{(\mathbf{p}_{ls}-\mathbf{p}_t)\times(\mathbf{p}_{rs}-\mathbf{p}_t)}{\|(\mathbf{p}_{ls}-\mathbf{p}_t)\times(\mathbf{p}_{rs}-\mathbf{p}_t)\|} \\ \mathbf{x} = \frac{\mathbf{p}_{ls}-\mathbf{p}_{rs}}{\|\mathbf{p}_{ls}-\mathbf{p}_{rs}\|} \\ \mathbf{y} = \frac{\mathbf{z}\times\mathbf{x}}{\|\mathbf{z}\times\mathbf{x}\|} \end{cases}$$

We then define the reference directions for involved joints (such as left arm direction, left fore arm direction etc) based on the coordinate system.

$$\mathbf{v}_{ref} = \alpha\mathbf{x} + \beta\mathbf{y} + \gamma\mathbf{z} \tag{1}$$

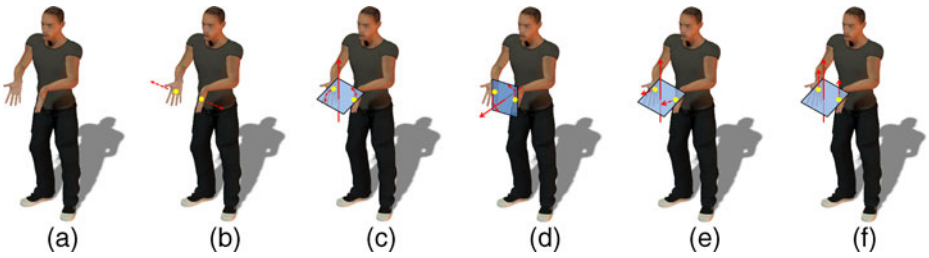

Fig. 5 Assembly gestures. **a** Activating the assembly mode; **b–c** Rotation (*roll and yaw*); **d–e** Translation in z and y directions; **f** Releasing the block

In the navigation and assembly modes, we check whether the user has performed a certain gesture by comparing the current direction $\mathbf{v}_c$ and the reference direction $\mathbf{v}_r$. The gesture is accepted when

$$\mathbf{v}_c \cdot \mathbf{v}_r > \eta. \tag{2}$$

4 Snapping interface

As mentioned above, one of the key challenges in developing a Kinect based virtual assembly is the inaccurate depth information of the obtained 3D video. Further, we can not expect the users to preform these gestures in a very accurate manner, since the users do not have any tangible references in the real world. Due to these issues, it is very tedious for the user to accurately place the block at the desired location.

To tackle this challenge, we further propose a snapping interface, which significantly reduces the user's burden to accurately place the object at the desired location. When the avatar approaches the already-assembled blocks, our system *automatically* computes the optimal location and orientation of the to-be-assembled block. This is done by searching in the nearby region of interest with various criteria, such as distance, alignment, contact, balance, and symmetry, which are formulated into a nonlinear optimization problem. Note that this optimal location depends on the avatar's current position. When the avatar moves, our system automatically updates the suggested location and orientation of the to-be-assembled block. If the user is satisfied with the suggestion, he simply releases the block by a releasing gesture.

In the following, we first describe the technique to analyze the geometry of the building blocks to get the base, side and supporting faces. Then we format our snapping interface into a nonlinear optimization problem. Finally we detail the technique to solve the optimization problem efficiently.

4.1 Shape analysis

Building blocks usually have simple geometry (such as square, cylinder, arch, triangle, cone, pyramid, etc) which provides important assembly constraints to the others. We follow Mitra et al.'s approach [12] to extract the features/attributes of the building block and segment it into primitive shapes if necessary. Then, we partition the boundary surface of the blocks into three categories:

1. **Base faces** are the faces on the *current to-be-assembled block* whose normals point to the down direction. See the green faces in Fig. 6b. To visualize the base faces, we intentionally rotate the block in this figure.)
2. **Supporting faces** are the faces on the *already-assembled blocks* whose normals point to the up direction. See the red faces in Fig. 6b.
3. **Side faces** are the faces on *all the blocks* whose normals are perpendicular to the up/down direction. See the blue faces in Fig. 6b.

Note that when the user changes the orientation of the current to-be-assembled block, our system automatically updates its base and side faces.

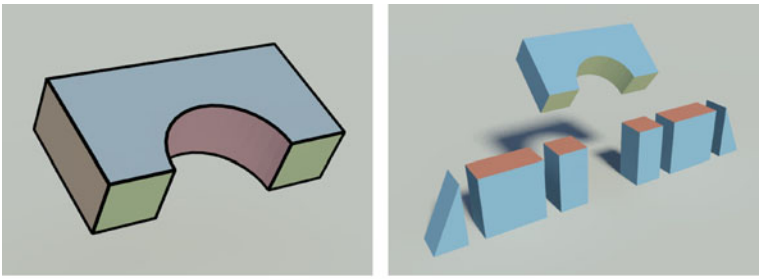

Fig. 6 Shape analysis. **a** Detected feature curves are used to partition the boundary surface of the to-be-assembled block. **b** Base (*green*), supporting (*red*) and side (*blue*) faces

4.2 Snapping criteria

Our interface is able to snap the block onto the optimal position that is close to the avatar's current location. To design the snapping criteria, we observe that the following rules are usually obeyed in a real-world building block assembly scenario:

- **Supporting rule**. There is at least one supporting face with the normal consistent to the up direction, and the block's center of mass must be within the supporting area formed by one or multiple supporting blocks. As shown in Fig. 7, the assembly in (b) is stable, but the one in (a) is not, due to the lack of the supporting faces.
- **Order rule**. User can assembly a new building block only when all the supporting blocks are ready. For example, we will not assembly the table top until all the four legs are ready (as shown in Fig. 7b).

With the current to-be-assembled block carried by the avatar, we check all the already-assembled blocks that are within distance r of the avatar (see the region of interest in Fig. 2c), then find the corresponding supporting faces. Then we compute

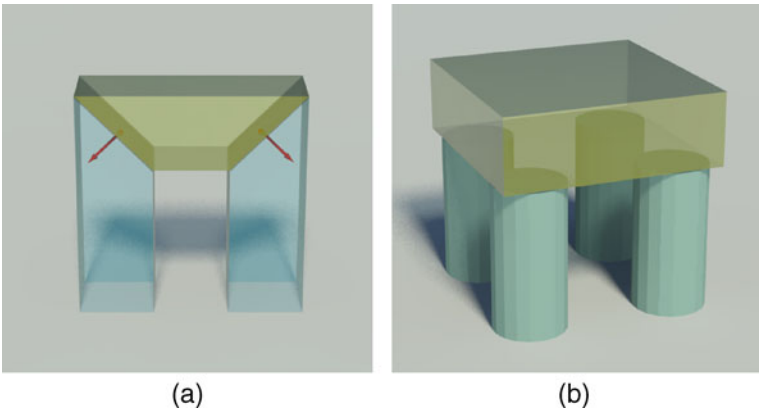

Fig. 7 Assembly rules. **a** Support rule; **b** Order rule

an optimal transformation which can align the base surface of the current block to those supporting faces perfectly. We formulate it as an optimization problem by considering the following five factors. To better balance among these factors, we normalize the energy formulas, so that each term falls in 0.0 and 1.0.

Distance This energy term penalizes the distance from the avatar's current location to the suggested location that the block will be placed, which results in placing the block close to the avatar. The distance term is defined as:

$$E_d = \| \mathbf{c}_b - \mathbf{c}_s \|^2 / (r^2 + h^2) \tag{3}$$

where $\mathbf{c}_b$ is the current block center position, $\mathbf{c}_s$ is the suggested location, h is the maximum hight of blocks, and r is the radius of the region of interest.

Alignment is an important principle that has been extensively employed in assembly tasks. For building block assembly, the alignment term checks both the orientation of the blocks and the distance between feature curves of the to-be-assembled block and the already-assembled blocks, which share the same orientation. We collect all the feature curves of the supporting faces (denoted by c_i^s) that are matched to the feature curves of the base face (denoted by c_i^b). The alignment energy is defined as:

$$E_a = 1.0 - \frac{\sum_i l_i^b \sum_{j \in C_i^s} e^{-2d(c_i^b, c_j^s)} l_j^s}{\sum_i l_i^b \sum_j l_j^s} \tag{4}$$

where l_i^b is the length of curve c_i^b, l_j^s is the length of curve c_j^s, and $d(c_i^b, c_j^s)$ measures the distance between two curves.

Contact The contact term measures the contacting area between the current block and the already-assembled ones. The larger the contacting area, the better the configuration of the assembly. We consider the contact area of both the base faces and side faces. For the former, we project the base faces and the supporting faces onto the horizontal plane (e.g. the ground), and compute the intersection polygons between the projected base faces and supporting faces. The area of the side contact is computed in a similar manner. Let A_{ib} (resp. A_{is}) denote the area of the intersection polygon and A_b (resp. A_s) be the area of the base (resp. side) faces. Then, the contact energy is given by:

$$E_c = 1.0 - \left[w \sum A_{ib} / \sum A_b + (1 - w) \sum A_{is} / \sum A_s \right] \tag{5}$$

We set the weight $w = 0.7$ in our implementation.

Balance is very important in building block assembly. Without balance constraint, the building block could collapse easily. Let $\mathbf{c}_s$ be the supporting center, computed by the weighted sum of the center of each intersect supporting polygon between base faces and supporting faces:

$$\mathbf{c}_s = \sum_k A_i^k \mathbf{c}_i^k, \tag{6}$$

where A_i^k is the area weight. Then we define the balance energy term to ensure the mass center of the current block $P(\mathbf{c}_b)$ is as close as possible to the supporting center:

$$E_b = \| P(\mathbf{c}_b) - \mathbf{c}_s \|^2 / d_s^2, \tag{7}$$

where d_s is the diagonal length of the bounding rectangle of those projected intersect supporting polygons.

Symmetry is ubiquitous for man-mad objects. We currently consider two kinds of symmetries: global and local reflective symmetry. The arrangement of two pink blocks in Fig. 8 shows an example of global reflective symmetry, and the two orange blocks on the green square is a local reflective symmetry.

The global reflexive axis is specified by the user before the virtual assembly session, e.g. the avatar follows the marked lines on the ground, which are used as the reflexive axes.

Let B_c be the current to-be-assembled block and B_r be its reflective image with respect to the global reflexive axis. To calculate the global symmetry energy, we search those already-assembled blocks, and find the block (denoted by B_s), which has the same shape as B_c and is within a distance d of B_r ($d = 0.1$ in our implementation). Let $\mathbf{p}_i^r$ be a feature point (e.g. a corner of the square, or sample points on a feature curve, etc) on B_r and $\mathbf{p}_i^s$ be the feature point on B_s that is closest to the mirror image of $\mathbf{p}_i^r$.

We then compute the matching criteria m between B_r and B_s as follows:

$$m = \sum_i d(\mathbf{p}_i^r, \mathbf{p}_i^s) \cdot (2.0 - \mathbf{n}_i^r \cdot \mathbf{n}_i^r) \tag{8}$$

where $\mathbf{n}_i^r$ and $\mathbf{n}_i^r$ are the corresponding normal of the two feature points. Measuring the normal variation in (8) can guarantee the matching on the *correct* sides.

We compute the matching criteria for each reflection axis in the global setting and obtain the global symmetry energy E_g:

$$E_g = 1.0 - \frac{1}{n} \sum_{k=1}^{n} e^{-2m_k}, \tag{9}$$

where n is the total number of global reflection axis.

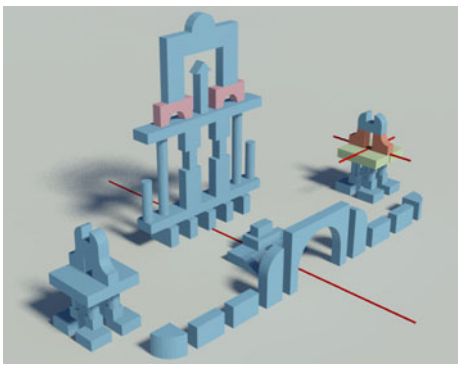

Fig. 8 Symmetry is ubiquitous in building block game

The local symmetry energy E_l is computed in a similar fashion, except the symmetry axis is detected automatically from each symmetry block of the already-assembled ones.

The total symmetry energy is given by the weighted sum between the global and local symmetry energy:

$$E_s = w\,E_g + (1 - w)\,E_l.$$

The weight w is set 0.6, i.e., with more focus on the global symmetry.

Putting it together, we define the objective function for our snapping interface as the weighted sum of the above energy terms:

$$F(\mathbf{p}, \theta) = w_d E_d + w_a E_a + w_c E_c + w_b E_b + w_s E_s \tag{10}$$

In our implementation, the weights are set empirically. To be specific, we set $w_d = 1.0$, $w_a = 3.0$, $w_c = 1.2$, $w_b = 1.2$, and $w_s = 1.5$ which give the alignment the highest priority.

4.3 Computig optimal solution

The objective function (10) is nonlinear and the searching space is highly complex, which makes it difficult to solve. One possible approach is to use stochastic optimization methods wich have been used in graphics community to solve furniture or architectural layout problem [7, 8, 27]. However, the Metropolis-Hasting algorithm is computationally expensive and is not suitable for interactive applications like ours. Parallel implementation [8] can improve the performance, but the implementation is nontrivial.

Observe that building block assembly follows many constraints (e.g. balance, alignment, etc), which can be used to significantly reduce the solution space. In this paper, we adopt a different yet effective strategy to tackle this nonlinear optimization problem by fully exploiting the various constraints. First, we find the **combination of nearby supporting faces** of the already-assembled blocks, which can provide a *valid* support of the to-be-assembled block. By doing this, we limit the search space in several combinations of nearby supporting faces avoiding the search over the whole scene; Then, we use an effective **filtering** to remove the invalid or poorly-matched orientations to further limit the search space; Finally, we conduct a **hierarchy-based stochastic search** to find the best candidate position for each valid orientation. We choose the one with the least energy as the optimal solution among all the candidates.

4.3.1 Checking supporting faces

We collect the *available*[2] supporting faces and side faces in the region of interest. We call these faces the involved supporting faces and side faces. Note that the to-be-assembled block may contain multiple base faces which are not co-planar (e.g. a T-shaped block). We enumerate all the possible combinations of base faces and

[2]We mean the face is not fully contacted by other faces.

check the validity of each combination. A base face combination is valid if it satisfies the following two conditions:

1. **Balance**. The barycenter of the to-be-assembled block is above the supporting polygon defined by the base faces.
2. **Support**. There is a matched supporting face for each base face, so that the base faces can be placed on top of the supporting blocks.

For each valid combination of base faces, we find all the matched supporting faces. We only consider those supporting faces in the avatar's region of interest.

4.3.2 Orientation filtering

We discretize $\theta \in [0, 360°]$ in (10) with an increment of $\triangle\theta = 5°$. For each orientation, we check whether the normals of the base group are consistent with that of the supporting group. Next, we measure the validity of the orientation by:

$$C_o = 1.0 - \left[w \frac{\sum_i l_i^b \sum_{j \in C_i^s} l_j^s}{\sum_i l_i^b \sum_j l_j^s} + (1 - w) \frac{\sum A_{is}}{\sum A_s} \right] \tag{11}$$

Note that the above equation is similar to the alignment energy term in (4) except that the distance weight is removed. We compute the average of the above measures for all orientations that pass through the normal consistency test. Finally, we discard the orientations whose measures are above the average. In practice, this filter is very effective to eliminate almost 80 % of the orientations.

4.3.3 Hierarchy based position optimization

After filtering out the invalid orientations, we search the optimal translation $\mathbf{p}$ for each orientation θ. The search space can be greatly reduced by exploitation of the constraints among blocks. As shown in Fig. 9, the supporting faces and side faces of the already-assembled blocks induce various constraints on the translation $\mathbf{p}$, such as the plane constraint in Fig. 9a and the line constraint in Fig. 9b. These constraints can be further combined into a new one. For example, the two plane constraints in Fig. 9c are combined into a line constraint, while the two line constraints in Fig. 9d is combined into a point constraint. To efficiently deal with these constraints, we build a constraint hierarchy from constraints induced from both the supporting groups and the side groups (see Fig. 10). The first level in the hierarchy is the combined constraints from the supporting groups. The second level consists those constraints which are combined from the side constraints and the first level constraints. Finally, the k-th level ($k \geq 2$) are constructed by merging each pair of constraints in the $k-1$ level. The construction of the hierarchy stops when the constraints in $k-1$ level cannot be further merged.

With the constraint hierarchy, we begin to search for the optimal snapping configuration from the top level to the bottom. We give higher priority to the higher levels because they may lead to more side contact and thus result in lower cost. We use stochastic optimization method to find the optimal position. Markov chain Monte Carlo sampler is employed to explore the cost functions of (10). We define a Boltzmann-like density function for each cost function

$$p(\mathbf{X}) = \frac{1}{Z} \exp(-\beta C(\mathbf{X})),$$

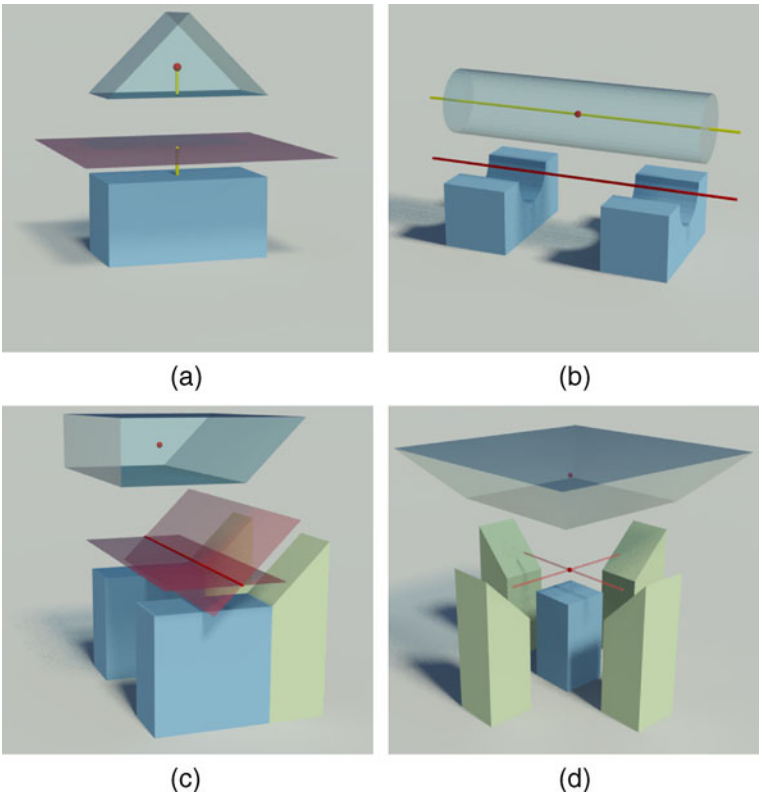

Fig. 9 Block constraints

where β is a temperature constant. Z is the partition function that normalizes the distribution. The Metropolis-Hastings algorithm [2, 9] is used to explore the density function. A current configuration X is maintained and new configuration X^* is proposed iteratively. The new configuration will be accepted as the current configuration with probability

$$\alpha(X \rightarrow X^*) = \min\left(1, \frac{p(X^*)q(X|X^*)}{p(X)q(X^*|X)}\right), \tag{12}$$

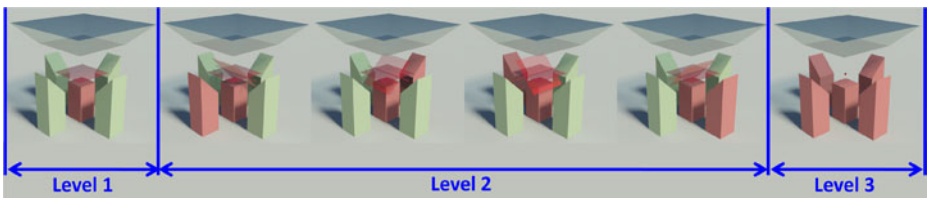

Fig. 10 Constraint hierarchy. The first level contains the constraints induced by the supporting faces (*the red block*). The second level consists of those by merging the side constraints with the supporting constraints. The higher levels of the constraint hierarchy are generated by merging pairs of the lower level constraints. The *red blocks* show the involved blocks in each constraint

where $q(X^*|X)$ is the proposal distribution from which a new configuration X^* is sampled given a current configuration X. Similar to [8], the proposal distribution choose between both local and global proposal moves to rapidly explore the density function. The proposal moves are as follows:

- Local proposal: Pertub the position of the block by adding a Gaussian term $N(0, \sigma^2)$ to each coordinate.
- Global Proposal: Swap the x, y component of the block position.

The standard deviation σ and σ_θ are set empirically. As the proposal distribution is symmetric, that is, $q(\mathbf{X}^*|\mathbf{X}) = q(\mathbf{X}|\mathbf{X}^*)$. Thus the acceptance probability (12) could be reduced to the Metropolis ratio, which can be computed directly from the cost function c:

$$\alpha(X \rightarrow X^*) = \min\left(1, \frac{p(X^*)}{p(X)}\right). \tag{13}$$

With all the candidate solutions, i.e., the translation $\mathbf{p}$ obtained by the stochastic searching for each valid orientation θ, we evaluate the cost function $F(\mathbf{p}, \theta)$ in (10) and find the one with the least energy as the optimal solution.

5 User study

We conducted a user study to evaluate our computer-aided block building system. There are two main purposes in the user study: (1) to validate that the kinect-based interaction is more easy to learn and use than direct manipulation style; (2) to validate that the snap interface is helpful for quickly assembly of a block into building scenario. 24 participants were recruited in our user study, 14 males and 10 females. Among them, 12 are children(aged between 8 and 12), four are adults(aged between 35 and 48) and the others are computer science students(aged between 20 and 22). The user study consists of three stages: in the first stage, we would like to collect and analysis necessary user data to determine the appropriate parameter values for gesture identification (1) and (2); the second stage will compare the kinect-based interaction with direct manipulation style; the third stage will explore the usability of the snap interface in block assembly process by comparison of assembly between with and without snapping. During the second and the third experiments, the participants are uniformly divided into two groups to perform the tasks in each experiment in reverse orders. The purpose is to minimize the potential discrepancy due to familiarity with the system.

Experiment # 1 In this experiment, the participants are asked to perform each of the gestures in the gesture sets three times. We can then calculate the intrinsic parameters α, β and γ in (1) based on the captured skeleton data for each user each time. Figure 11 shows the captured data for the right fore arm limb of the holding posture. For comparison purpose, we also calculate the corresponding parameters under the default coordinate system defined by the depth camera. We can see that the intrinsic parameters are more stable among different participants which is further validated by the standard deviation of the parameters in Fig. 11b.

As the variation of the intrinsic parameters are very small, we thus define the reference direction of each limb under the specific pose using (1) with the overall

	Standard Deviation					
	All		Children		Adults	
	I*	D*	I*	D*	I*	D*
α	0.073	0.096	0.081	0.105	0.062	0.096
β	0.135	0.154	0.153	0.155	0.120	0.163
γ	0.027	0.032	0.027	0.050	0.026	0.030
η	0.031	0.034	0.037	0.038	0.021	0.031

*: "I" means intrinsic, "D" means default.

(a) (b)

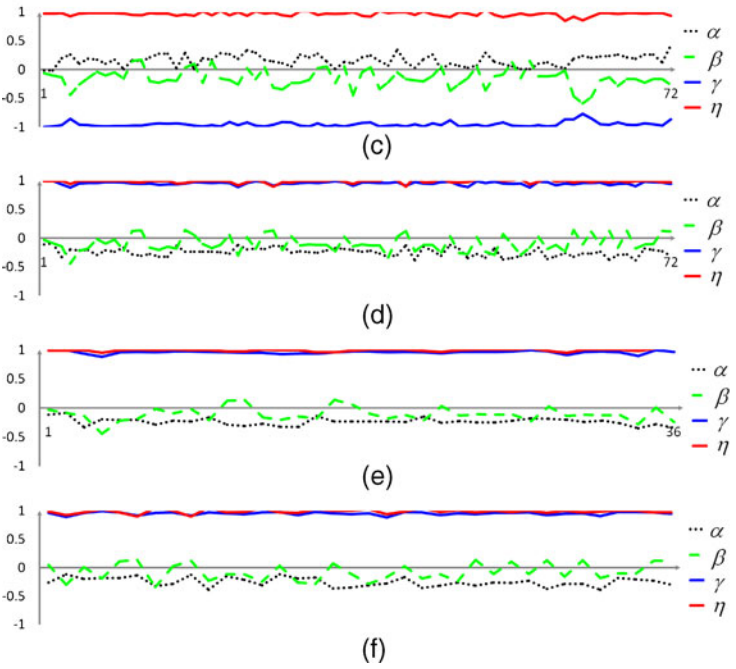

Fig. 11 User study experiment # 1. The plots show parameters of the right fore arm in the holding posture (**a**) calculated under the default coordinate system for all the participants (**c**) and under the intrinsic setting for all (**d**), adults only (**e**) and children only (**f**) respectively. The table on the *top right hand side* (**b**) shows the variation of the parameters in intrinsic and default settings

averaged triple. With the reference direction, we can then evaluate the difference between each example direction of the limb and the reference by calculating the dot product of the two vector as in (2). We finally set the threshold value η for each limb under each pose as the maximum one.

Experiment # 2 The purpose of the second experiment is to compare the kinect-based interaction with traditional direct manipulation style to see whether the former is more easy to learn. To be more specific, we want to explore how quickly can user learn the mapping between the gestures/keys and the related operations. According

to learning curve theory [25], repetition of the same operation results in less time or effort expended on that operation. The steeper the learning curve, the more easy it is to learn something.

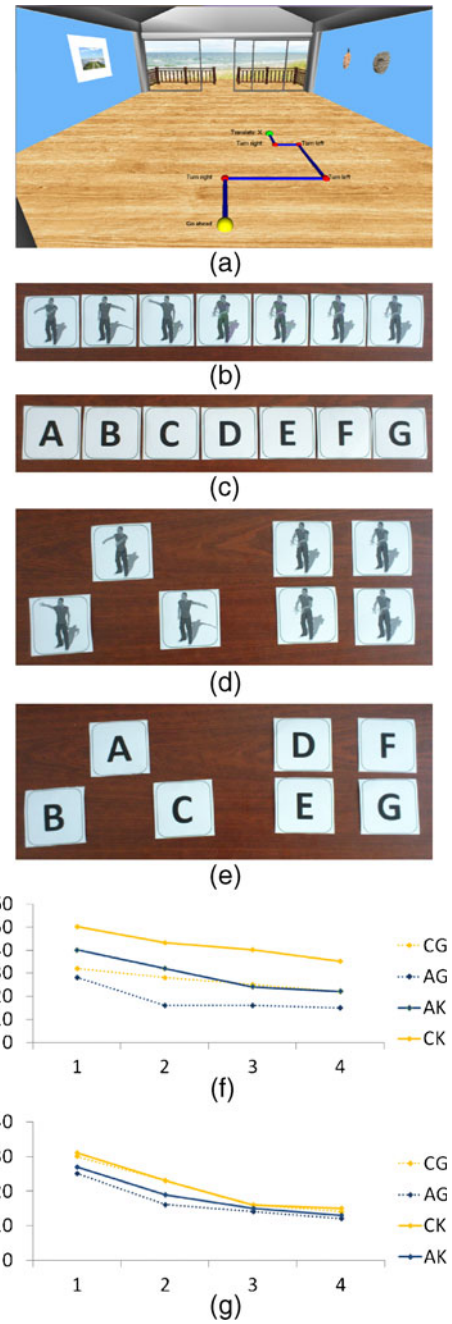

Fig. 12 User study experiment # 2. Given the task in (**a**), the participant was asked to use unarranged (**b**, **c**) or arranged tags (**d**, **e**) to specify the operation sequence for the task. **f** and **g** show the learning curves in arranged and unarranged settings respectively ('CG' means 'children using gesture', 'CK' means 'children using keyboard', 'AG' means 'adults using gesture', 'AK' means 'adults using keyboard')

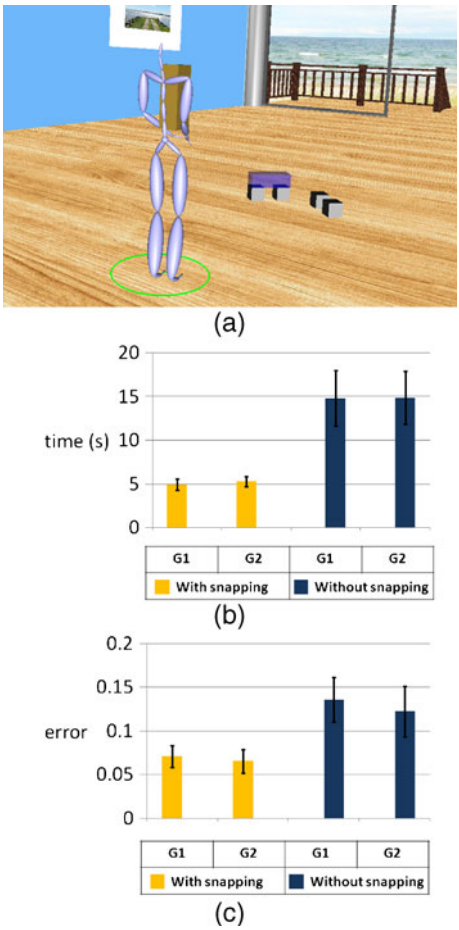

Fig. 13 User study experiment # 3. **a** Participant was asked to assemble the current block to the target position and orientation specified by the *blue block*. **b** Time spent; **c** Matching error; The error bars in (**b**, **c**) show for the 95 % confidence interval

The key issue in this experiment is how to measure the response time the participant taken to make the correct interaction (gesture/key) for a specific operation. Simply measure the time participant taken to select a block and put it onto the destination in the scene for both style is unfair. Due to anatomical constraints (making a gesture takes more time than simply pressing a key), it usually takes more time for kinect-based interaction to complete an assembly task. We carefully design a procedure to measure the response time more exactly. A picture showing the path to the destination is presented to the participant (Fig. 12a). Then he(or she) was asked to pick tags from gesture/key tag set to form the operation sequence for the assembly

Table 1 Paired t test results of experiment # 3

	Time		Error	
	G1	G2	G1	G2
DOF	11	11	11	11
t	7.31	7.32	6.08	6.52
p	<0.0001	<0.0001	<0.0001	<0.0001

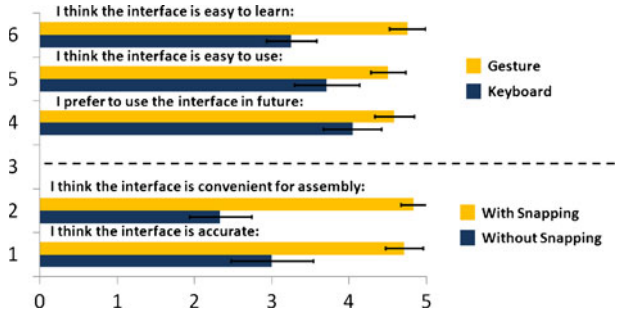

Fig. 14 Rating of our system on a scale between 1 and 5. (1 = strongly disagree, 5 = strongly agree), the error bars show the 95 % confidence intervals

task. The walk path was randomly generated for each style in each time, which meant the operation sequences needed to complete the task were different. We start timing when the picture is presented to the participant and stop when he (or she) finish the combination of tags. This procedure can avoid the interference of other factors such as anatomical constraints as the participant do the same action (pick a tag) for both styles. We also noticed that people usually like to defined the keys for the instructions in a specific pattern on the keyboard so as to help them remember the mapping between the keys and the instructions easily. Thus we separates the participants into two teams with one of the team can arrange the tags in any pattern they are most comfortable (Fig. 12d, e), while the other team is not allowed to rearrange the tags (Fig. 12b, c). Each team is further divided into two groups, one group using body gesture first and then keyboard, and the other group perform in the reverse order.

The timing statistics of the experiment are shown in Fig. 12f, g. We can clearly see that the trend of time cost and the reference count during the experiment meet the learning curve theory well for both interaction style. However, the curves corresponding to kinect-based interaction style are more steeper than the corresponding one of the direct manipulation style indicating that the kinect-based style is more easy to learn. Besides, we can see that it takes less time to finish the job with the kinect-based interaction style especially in the later session which validates that the kinect-based style is more easy to use.

Experiment # 3 In the third experiment, we would like to see whether our snapping interface can help participant to quickly assemble the holding block into the building scenario with desired position and orientation. The participant is asked to put the holding block into the scene with the target position and orientation specified by the blue one (Fig. 13a). Similarly, the participant need to perform the task twice with the snapping function switch on/off. The first group assembled the block with the snapping interface followed by without snapping interface, and the second group did in the reverse order. When the snapping function is switched off, the block was considered to be assembled appropriate if the matching error (defined as in (8)) $m(B, B_t) < \sigma$ ($\sigma = 0.3$ in our experiment).

Table 2 Wilcoxon signed rank test results of the questionaire

Question	1	2	3	4	5
Z value	3.97	3.02	2.00	4.28	3.81
P value	0.0001	0.0013	0.0228	<0.0001	0.0001

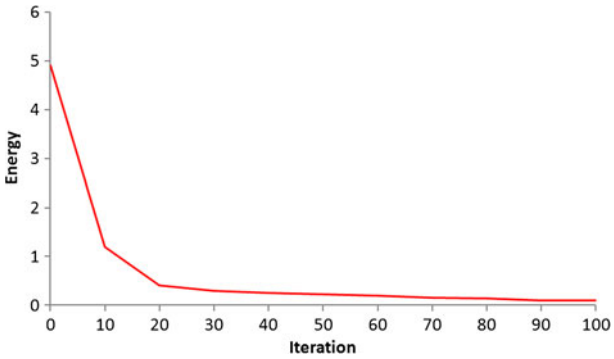

Fig. 15 Convergence of the snapping algorithm

From the statistics shown in Fig. 13b, c, we can see that the snapping interface can save about 60 % time in assembly of the block and with 45 % less error. We conducted the paired t-test on the timing and error data. From the statistics in Table 1, we can conclude that the snapping interface does significantly (with $p<0.05$) help to assembly the block more convenient and more accurate.

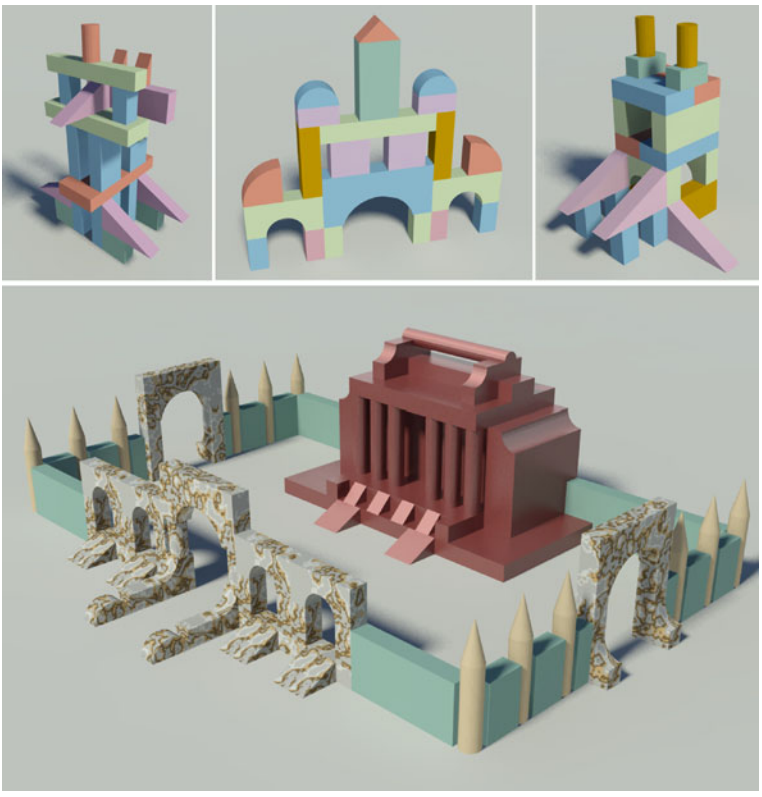

Fig. 16 Examples of the 3D virtual structures constructed by using our framework. Images are rendered using 3DS Max

Rating After completion the required tasks, the participants were asked to rate the body gesture and the keyboard on their 'easy to learn', 'easy to use' and 'preference' on a scale of 1 to 5: 1 means 'strongly disagree' and 5 means 'strongly agree'. Besides, they were also asked to rate the snapping interface on 'convenience' and 'accuracy'. Both the body gesture and the snapping interface received very positive feedbacks as shown in Fig. 14. We perform Wilcoxon signed rank tests on the questionnaire data to evaluate the significant effects of the availability of our interface on the listed aspects. Table 2 shows the test result. Users rated significantly higher on the 'easy to learn', 'easy to use' of the whole system and the 'convenience' and 'accuracy' of snapping interface($p<0.01$), but less satisfied with preference' on the system($p=0.0228>0.05$).

The whole system runs smoothly on a laptop with standard configuration (Intel Core Duo CPU 2.4 GHZ, 4 G RAM). It takes about 0.3 s for the snapping algorithm to find the optimal position for each assembly operation. Figure 15 shows the convergence of the algorithm when assembling the arc block to two cuboid blocks in Fig. 2c. The algorithm converges soon within 20 iterations. Figure 16 shows more examples of the building blocks constructed by our system.

6 Conclusion

This paper investigates the application of Kinect as a 3D interaction tool for toy block assembly. We design a set of intuitive body gestures that allow the user to naturally control 3D building blocks and navigate them in a virtual environment. To tackle the imprecise issue of the interactions with Kinect, we propose a snapping interface, which can automatically compute the optimal location and orientation of

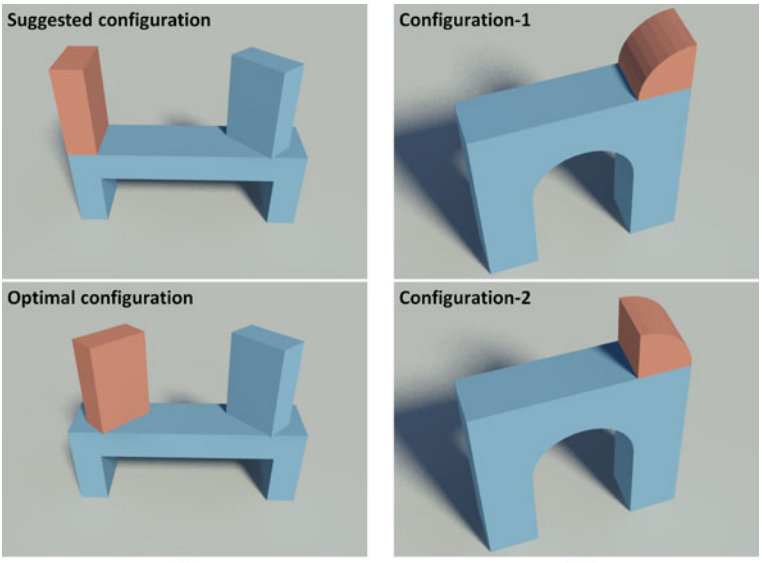

Fig. 17 Limitations. **a** Orientation blindness. **b** Orientation ambiguity

the to-be-assembled block. Thus, it not only reduces the user's burden for assembling the blocks at the desired locations but also allows the user to construct a complicated structure easily. We conducted a user study to evaluate the performance of the proposed snapping interface. The experimental results and the positive feedback from the participants demonstrate the efficacy of our snapping interface to virtual assembly of building blocks.

Due to the unstable nature of the captured depth video, Kinect SDK sometimes may fail in reconstruction of the skeleton, which in turn causes problems in our gesture analysis. For the snapping interface, our current implementation cannot find the optimal configuration for the highlighted red block in Fig. 17a. This is because we do not consider symmetry in the orientation filtering step, thus it is possible that the optimal orientation is filtered out. We call this *orientation blindness*. Also, our algorithm cannot eliminate *orientation ambiguity*. For example, it can not differentiate the two possible configurations for the red block as shown in Fig. 17b. A feasible solution is to provide all the possible configurations to the user who will make the decision.

Acknowledgements This work was supported by the National Natural Science Foundation of China (No. 61202142, No. 61100032), Joint Funds of the Ministry of Education of China and China Mobile (No. MCM20122081), the National Key Technology R&D Program Foundation of China (No. 2013BAH44F00), the Open Project Program of the State Key Lab of CAD&CG Zhejiang University (No. A1205) and the Fundamental Research Funds for the Central Universities (No. 2010121072, No. 2013121030). AcRF 69/07, Singapore NRF Interactive Digital Media R&D Program under research grant NRF2008IDM-IDM004-006.

References

1. Fisher D, Hartmann B (2011) Deep window—3d virtual camera control with a tablet and depth camera. In: ACM SIGCHI
2. Hastings WK (1970) Monte carlo sampling methods using markov chains and their applications. Biometrika 57(1):97–109
3. Izadi S, Kim D, Hilliges O, Molyneaux D, Newcombe R, Kohli P, Shotton J, Hodges S, Freeman D, Davison A, Fitzgibbon A (2011) 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, pp 559–568
4. Kim YM, Mitra NJ, Yan D-M, Guibas L (2012) Acquiring 3d indoor environments with variability and repetition. ACM Trans Graph 31(6):Article No 137
5. Kin K, Miller T, Bollensdorff B, DeRose T, Hartmann B, Agrawala M (2011) Eden: a professional multitouch tool for constructing virtual organic environments. In: Proceedings of ACM SIGCHI, pp 1343–1352
6. Li X-Y, Shen C-H, Huang S-S, Ju T, Hu S-M (2011) Popup: automatic paper architectures from 3d models. ACM Trans Graph 29(4):Article No 111
7. Merrell P, Schkufza E, Koltun V (2010) Computer-generated residential building layouts. ACM Trans Graph 29(6):Article No 181
8. Merrel P, Schkufza E, Li Z, Agrawala M, Koltun V (2011) Interactive furniture layout using interior design guidelines. ACM Trans Graph 30(4):Article No 87
9. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092
10. Mitani J, Suzuki H (2004) Making papercraft toys from meshes using strip-based approximate unfolding. ACM Trans Graph 23(3):259–263
11. Mitra NJ, Pauly M (2009) Shadow art. ACM Trans Graph 28(5):Article No. 156
12. Mitra NJ, Yang Y-L, Yan D-M, Li W, Agrawala M (2010) Illustrating how mechanical assemblies work. ACM Trans Graph 29(4):Article No: 58

13. Mori Y, Igarashi T (2007) Plushie: an interactive design system for plush toys. ACM Trans Graph 26(3):Article No. 45
14. Nan L, Xie K, Sharf A (2012) A search-classify approach for cluttered indoor scene understanding. ACM Trans Graph 31(6):Article No 11
15. Preece J, Rogers Y, Sharp H (2002) Interaction design. Wiley, New York
16. Provenzo EF, Brett A (1984) The complete block book. Syracuse Univ Pr (Sd)
17. Shao T, Xu W, Zhou K, Wang J, Li D, Guo B (2012) An interactive approach to semantic modeling of indoor scenes with an rgbd camera. In: ACM SIGGRAPH Asia 2012. ACM Trans Graph 31(6):Article No. 136
18. Shen C-H, Fu H, Chen K, Hu S-M (2012) Structure recovery by part assembly. ACM Trans Graph 31(6):Article No 180
19. Shon Y, McMains S (2004) Evaluation of drawing on 3d surfaces with haptics. IEEE Comput Graph Appl 24(6):40–50
20. Tong J, Zhou J, Liu L, Pan Z, Yan H (2012) Scanning 3d full human bodies using kinects. IEEE Trans Vis Comput Graph 18(4):643–650
21. Ware C, Rose J (1999) Rotating virtual objects with real handles. ACM Trans Comput Hum Interact 6(2):162–180
22. Weise T, Bouaziz S, Li H, Pauly M (2011) Realtime performance-based facial animation. ACM Trans Graph 30(4):Article No 77
23. Weiss A, Hirshberg D, Black MJ (2011) Home 3d body scans from noisy image and range data. In: Proceedings of the 13th international conference on computer vision, pp 1951–1958
24. Wilson AD (2010) Using a depth camera as a touch sensor. In: ACM international conference on interactive tabletops and surfaces
25. Wright TP (1936) Factors affecting the cost of airplanes. J Aeronaut Sci 3(4):122–128
26. Xin S, Lai C-F, Fu C-W, Wong T-T, He Y, Cohen-Or D (2011) Making burr puzzles from 3d models. ACM Trans Graph 30(4):Article No. 97
27. Yu L-F, Yeung S-K, Tang C-K, Terzopoulos D, Chan TF, Osher SJ (2011) Make it home: automatic optimization of furniture arrangement. ACM Trans Graph 30(4):Article 86

Juncong Lin is a research fellow at NTU, Singapore. He received the Ph.D. degree in Computer Science from Zhejiang University. His research interests include geometry processing and sketch based modeling.

Qian Sun is a Phd candidate at NTU, Singapore. Her research interests include geometry processing and user interaction.

Guilin Li is now a lecturer at Xiamen University, P.R.China. He received the Ph.D. degree in Computer Science from Harbin Institute of Technology. His research interests includes CPS, distributed computing.

Ying He received the BS and MS degrees in Electrical Engineering from Tsinghua University, China, and the PhD degree in Computer Science from the State University of New York (SUNY), Stony Brook. He is an associate professor at NTU, Singapore. His research interests fall in the broad area of visual computing.