A Sketching Interface for Sitting Pose Design in the Virtual Environment

Juncong Lin, Takeo Igarashi, Jun Mitani, Minghong Liao, and Ying He, *Member*, *IEEE*

Abstract—Character pose design is one of the most fundamental processes in computer graphics authoring. Although there are many research efforts in this field, most existing design tools consider only character body structure, rather than its interaction with the environment. This paper presents an intuitive sketching interface that allows the user to interactively place a 3D human character in a sitting position on a chair. Within our framework, the user sketches the target pose as a 2D stick figure and attaches the selected joints to the environment (e.g., the feet on the ground) with a pin tool. As reconstructing the 3D pose from a 2D stick figure is an ill-posed problem due to many possible solutions, the key idea in our paper is to reduce solution space by considering the interaction between the character and environment and adding physics constraints, such as balance and collision. Further, we formulated this reconstruction into a nonlinear optimization problem and solved it via the genetic algorithm (GA) and the quasi-Newton solver. With the GPU implementation, our system is able to generate the physically correct and visually pleasing pose at an interactive speed. The promising experimental results and user study demonstrates the efficacy of our method.

Index Terms—Sketching interface, sitting pose design, virtual environment, genetic algorithm, quasi-Newton solver, GPU

1 Introduction

Designing a character pose is one of the most basic processes in computer graphics authoring. Although some automatic methods are available today, many artists still prefer to manually design the character pose by using direct control of joint angles and inverse kinematics (IK). With the existing pose design systems, users are often required to interactively position the extreme joints of a character and then the interior joints are automatically updated via inverse kinematics. Such an interface is powerful in that users can precisely specify the location of joints. However, manually positioning joints could be very tedious because of the difficulty in rotating the camera for proper viewing during the positioning; this situation can be even more difficult when the characters are in a complicated environment (e.g., those with occluded joint(s) that cannot be adequately revealed from many different viewing directions).

- *J. Lin is with the Software School of Xiamen University and the School of Computer Engineering, Nanyang Technological University, Singapore, Room A503C, Administrative Building, Haiyun Campus, Xiamen University, Fujian 361005, China. E-mail: jclin@xmu.edu.cn.*
- *T. Igarashi is with the Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Room 303, Science bldg. 7, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan. E-mail: takeo@acm.org.*
- *J. Mitani is with the Department of Computer Science, Graduate School of System Information Engineering, University of Tsukuba, Tennoudai 1-1-1, Tsukuba, Ibaraki 305-8573, Japan. E-mail: mitani@cs.tsukuba.ac.jp.*
- *M. Liao is with the Software School of Xiamen University, Room A201, Administrative Building, Haiyun Campus, Xiamen University, Fujian 361005, P.R. China. E-mail: liao@xmu.edu.cn.*
- *Y. He is with the School of Computer Engineering, Nanyang Technological University, N4-02a-22, 50 Nanyang Avenue, Singapore 639798. E-mail: yhe@ntu.edu.sg.*

Manuscript received 31 Jan. 2011; revised 23 Aug. 2011; accepted 16 Jan. 2012; published online 13 Feb. 2012.
Recommended for acceptance by R. Balakrishnan.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2011-01-0019.
Digital Object Identifier no. 10.1109/TVCG.2012.61.

To overcome the limitations in the existing pose design systems, this paper presents an intuitive sketching interface that allows the users to quickly and easily set a character's *sitting pose* while considering the interaction with the environment. Our primary target application is furniture design, in which the designers usually need to examine ergonomic and structural concerns [1]. For example, when designing a chair, the height must be appropriate such that one's feet are on the ground. In addition, the chair must remain stable even if the person sits in an unusual position.

Compared to existing systems, our system is superior in that it allows both professional users and beginners to easily and quickly design various sitting poses in a complex environment. Within our system, the user first draws a stick figure on the screen space to specify the desired sitting pose. This stick figure consists of circular dots, representing joints, connected by lines. The user can further specify whether a joint is attached to the environment via the pin tool (e.g., the feet of a sitting character are usually pinned to the floor). Our system then reconstructs a 3D pose from the input sketches and environmental constraints. This 3D reconstruction is fully automatic at an interactive speed, which allows the users to freely update the stick figure and easily improve the results. Fig. 1 shows several sitting poses that were generated by our system.

The key technique of our system is an approach to reconstruct a collision-free 3D pose from a 2D input stick figure. Note that the 2D-to-3D reconstruction problem is ill posed because of many possible solutions. In this paper, we show that by exploiting interactions between the character and the chair, one can significantly reduce the solution space, thus, eliminate undesired poses. Specifically, we formulated the 2D-to-3D reconstruction as a nonlinear optimization problem and proposed a hybrid solver that consists of two components: a genetic algorithm-based solver (G-A solver) is used to compute a collision-free pose

Fig. 1. Various sitting poses designed by our system. The user sketches the 2D stick figures on the screen space to specify the desired poses in a virtual environment (see the inset of each subfigure), and our system automatically generates the physically correct and visually pleasing 3D poses at interactive speed.

and a quasi-Newton procedure (Q-N solver) that is applied to further refine the result. With the GPU implementation, our system can generate the physically correct and visually pleasing pose at an interactive speed.

The preliminary version has appeared in [2]. In this extended paper, we have incorporated several major changes that greatly improve the original system:

- We greatly improved the accuracy and efficiency of our pose reconstruction solver with two new techniques. First, we parallelize the GA solver onto GPU with CUDA. Note that quite a small population size was used in our previous system [2] to achieve interactive speed. Additionally, we observed that such a strategy affected the sampling ability of the solver and thus reduced its accuracy. By parallelizing the GA solver onto the GPU, we now can use a much larger population size while maintaining the interactive speed. Second, we adopted a new strategy to handle the collision detection issue. The previous system attempted to avoid the collision case using a trial-and-error method during the evolution process. In our new system, we gradually replaced those collision cases with collision-free cases in the main population during each iteration. As a result, we gain a speedup of 5 with the same population size.
- We observed that users have very different painting styles that may lead to large distortions in the sketched 2D stick figures, which, in turn, would significantly affect the reconstruction quality of our previous system. In our new system, we reduced such artifacts by normalizing the stick figure before pose reconstruction so the projected bone length is now in a reasonable range. As a result, our new system allows a wider range of users, including both beginners and experienced designers, to freely and efficiently design the desired poses in a complicated virtual environment. We also improved the user interface by adding some new features. For example, we provided an editing tool that allows the user to further edit the stick figure after drawing. We further improved the pin tool to allow user to specify the exact attach target when there are multiple possible attach positions. As seen in the second example of Fig. 1, the feet can be attached to either the table or the ground. The quantitative results of the user study and positive feedback from participants demonstrate the efficacy of our system in sitting pose design.

The remaining of the paper is organized as follows: Section 2 reviews the related works on inverse kinematics, sketching interfaces and 3D reconstruction. Section 3 presents the overview of the proposed sketching interface while Section 4 presents the technical details of our algorithm. Section 5 shows our experimental results, followed by the user study in Section 6. Finally, Section 7 discusses the limitations of our work and Section 8 draws the conclusion.

2 Related Work

Inverse kinematic [3], is a widely used technique for character pose design. In spite of its popularity, IK is known to be almost always underdetermined. Thus, lots of previous efforts [4], [5] have been made to restrict the solution space with various constraints that lead to feasible solutions. As discussed in Section 1, the inverse kinematics-based pose design systems often require the users to interactively position the extreme joints of a character. These positioning procedures may be awkward if the character is in a complicated environment, since the occluded joint(s) cannot be adequately revealed from many different viewing directions.

Traditionally, animators often began work by quickly sketching 2D stick figures in key poses to capture the character's overall motion. Inspired by this, several sketching interfaces have been developed to facilitate the user's input. For example, Davis et al. [6] reconstructed a 3D pose from an artist-drawn 2D stick figure and provided a complementary alternate for 3D pose design. Wei and Chai [7] formulated the interactive character posing problem via a maximum posteriori framework and also discussed direct manipulation interfaces and sketching interfaces. Jain et al. [8] proposed a method to generate 3D animation of human motion from traditional hand-drawn frames that leverage motion capture data as a source of domain knowledge. Hardware input device has also been proposed to intuitively control character pose [9]. Of note, we adopted a similar sketching interface as [6]. However, our method is different from previous work in two aspects: 1) we focused more on the interaction between the character and the environment and 2) we formulated the 3D pose reconstruction as an optimization problem with both sketching and physical constraints.

Our work is also closely related to the reconstruction of 3D poses from monocular images. Most existing works have focused on using various domain constraints to solve the

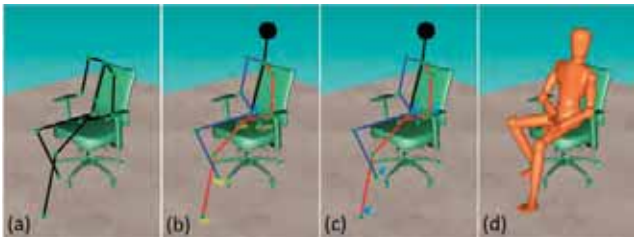

Fig. 2. Sitting pose design flow: the user roughly sketches a 2D stick figure on 2D screen space to represent the desired pose (a) and then specifies the left and right limbs to eliminate symmetrical ambiguity (b). Next, the user can attach the joints to the environment with the pinning tool (c). Finally, the 3D pose is generated automatically at interactive speed.

underconstrained depth ambiguity problem. These works can be roughly classified into either model-based technique [10], [11], [12] or probabilistic learning-based techniques [13]. However, our work is different than the previous methods in that 1) our method takes advantage of physical constraints and the interactions between the character and environment to eliminate inappropriate solutions and 2) our input is a hand-drawn stick figure, which is usually imprecise and needs special treatment.

3 User Interface

In a typical design session, the user roughly sketches a 2D stick figure on the screen space to represent the desired pose (see Fig. 2a). Note, the edges of the stick figure can be drawn in an arbitrary order. Our system then analyzes the stick figure and organizes it into a tree structure similar to the character's skeletons. After this organization, the user can check the tree structure and specify the left and right limbs to eliminate symmetrical ambiguity, if necessary[1] (see Fig. 2b). The user can also use the pinning tool to attach the joints to the environment (e.g., the feet on the ground, the hands on the armrest, and so on (see Fig. 2c). Finally, our system automatically generates the 3D pose at an interactive speed (see Fig. 2d).

3.1 Drawing Tool

We also developed a simple drawing tool similar to [6], in which the user draws straight limb segments by dragging, rather than freeform drawing; the user presses the mouse button to specify the position of the first joint then drags and releases the cursor at the desired position of the second joint. This process is repeated until the entire skeleton has been specified. A newly sketched limb will automatically snap to an existing one if its joint is close enough to the joints of the previous limb. This snapping function makes it easy for the user to ensure that segments are connected properly. Note that the user sketches elements of the pose in an arbitrary order and cannot change the view point during sketching. As a result, the stick figure is drawn purely on the 2D screen space.

3.2 Editing Tool

The user can freely edit the 2D stick figure in either local or global mode. Within the local mode, the user can drag any individual joint without changing other joints. The editing tool can also be automatically switched to the global model if a joint is dragged a certain distance, d, or further away from the original position ($d = 100$ pixels in our implementation), as shown in Fig. 8. Specifically, joints that are between the root joint and the dragging joint will be updated in an as-rigid-as-possible way so as to preserve their relative orientation as much as possible. Further, the children of the moving joints (the dragging joint and the affected intermediate joints) will rigidly move and follow their parents. However, if a joint is only modified slightly, the editing will be local. User can also manually switch between the two editing modes.

1. Red (resp. blue) edges represent the left (resp. right) side of the character.

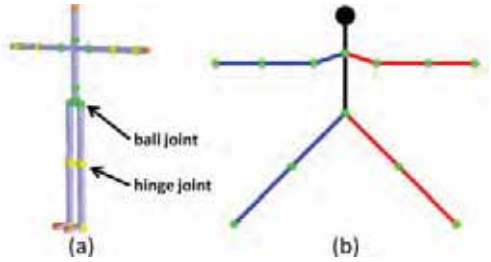

Fig. 3. The human model used in our system: (a) template skeleton of the 3D character; (b) template stick figure used to specify the pose.

3.3 Pinning Tool

The pinning tool is used to specify the target position in the environment to which a joint is attached (e.g., putting the feet on the ground). The pinning tool can also be used to attach a joint to a limb of the character. This is useful for the scenario of the character sitting with the hands on the legs. To pin a joint, the user simply clicks on the sketched node. If multiple attached candidates are detected, a dialog box will popup that allows the user to specify the desired candidate (e.g., the right feet in Fig. 1b can be attached to either the table or the ground). The system detected limb attachments by checking whether the pinning joint is close enough to a certain limb in the stick figure on the screen space. For environment attachments, we simply emitted an eye ray that passes through the pinning joint and finds all intersections within the environment. These intersection points are then added to the candidate list. Of note, intersection points can be points either on the horizontal floor or in the environment with various normal directions. The pelvis joint is an exception in that it can only be pinned to the environment (sitting plane).

4 Technical Details

4.1 Sketch Analysis

4.1.1 Human Model

We currently use a human model with 29 degrees of freedom (DOF) as shown in Fig. 3a. The human model is represented by a tree of joints ($H = \{\mathbf{j}_k\}_{k=1}^{m}, m = 20$) rooted at the pelvis. The stick figure (Fig. 3b) is used to specify the pose in the screen space. It is also organized in a simple tree structure with 13 nodes ($S = \{\mathbf{s}_i\}_{i=1}^{n}, n = 13$). Let $\Phi : \mathbf{s}_i \longrightarrow \mathbf{j}_k$ be the function that maps the stick nodes to the skeleton joints.

The user-drawn sketch is initially defined as an undirected graph. To construct a tree structure from the graph,

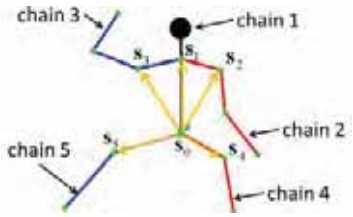

Fig. 4. Eliminating symmetric ambiguity in the stick figure.

the system first identified the pelvis (of degree 3) and neck (of degree 4), then extracted the chains from the root to the leaf nodes (of degree 1). Next, we distinguished the arms and legs by counting the number of nodes on the chains. For example, there were five nodes from the root (pelvis) to the left or right hand and three nodes from the root to the feet. To define the mapping Φ, we also must eliminate the symmetric ambiguity of the tree structure. Assuming the user draws a front view pose of the character, we then identify the left and right limbs by checking the following equation (Fig. 4):

$$sign = \mathbf{s}_0\mathbf{s}_i \cdot \mathbf{s}_0\mathbf{s}_1 (i \neq 0, 1).$$

Chain i represents a left limb when $sign > 0$, right limb, otherwise. Our interface also allows the users to manually switch left and right sides if the automatic determined orientation is wrong (see Fig. 2b).

4.1.2 Sit Location

To reconstruct a pose, the system first needs to know where the person sits. As shown in Fig. 5a, users may have very different sketching styles (e.g., some may draw the central line of each limb (red), while others may place the sketched joints at the contact between the character and the floor or chair (blue)). Clearly, the algorithm for locating the sitting position depends on the user's painting styles. Therefore, we conducted an observational study to better understand the way people draw. Specifically, we showed participants photos of sitting poses (see Fig. 5b for an example) and asked them to sketch 2D stick figures using our interface. Fig. 5c shows several typical sketches collected from the study. Observing that most users drew the skeletons as center lines, we adopted the center line strategy as the default sketching style in our implementation. Our system also allows the user to switch, manually, to a contact-point sketching style.

To find the sitting location, we calculate the intersection between the extended bounding box of the chair and the ray $l_e(\mathbf{p}_e, \mathbf{v}_e)$ from the eye to the pelvis node (see Fig. 6a). We

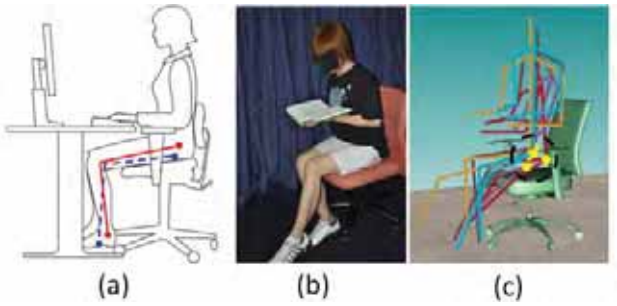

Fig. 5. Observation study. (a) Different sketching styles; (b) a sitting pose shown to the participants of the study; (c) several sketched poses.

extend the bounding box in an upward direction $\mathbf{v}_u = (0.0, 1.0, 0.0)$ to ensure the intersection with some object, such as the foot stool in Fig. 9. We then sample the intersection line segment in the bounding box and emit a ray from each sample in the direction $\mathbf{v}_s = -\mathbf{v}_u$ (Fig. 6b). We collect the intersected faces whose normal satisfies

$$\mathbf{n}_f \cdot \mathbf{v}_u > \sqrt{3}/2 \tag{1}$$

and where the distance to the sample point satisfies

$$s_1 \cdot h \leq d \leq s_2 \cdot h, \tag{2}$$

where h is the hip height, $s_1 = 0.5$ and $s_2 = 1.5$. The constants in (1) and (2) are chosen empirically by the users. Using these intersected faces as seeds, we can grow several regions by collecting all the faces for which the normal satisfies (1). For each region R_i, we assign a normal $\mathbf{n}_r$ by averaging the face normals over the whole region. We can then generate a cut plane P_i passing through the eye position $\mathbf{p}_e$ and the normal $\mathbf{n}_p = \mathbf{n}_r \times \mathbf{v}_e$. We calculate the uniformly resampled intersection curve $I_i = (\mathbf{p}_0, \mathbf{p}_1, \ldots \mathbf{p}_m)$ between P_i and R_i, and find the best sitting point of R_i by minimizing the following equation:

$$M(R_i) = \arg\min_k (|d(\mathbf{p}_k, l_e, \mathbf{n}_r)) - h| \cdot C(k, m)). \tag{3}$$

The metric consists of a distance term and a centerness term: the distance term measures the difference between $d(\mathbf{p}_k, l_e, \mathbf{n}_r)$ and the hip height h. $d(\mathbf{p}_k, l_e, \mathbf{n}_r)$ is the distance between $\mathbf{p}_k$ and its projection on the eye ray l_e along the normal direction $\mathbf{n}_r$

$$d(\mathbf{p}_k, l_e, \mathbf{n}_r) = \frac{\| (\mathbf{p}_k - \mathbf{p}_e) \times \mathbf{v}_e \|}{\| \mathbf{n}_r \times \mathbf{v}_e \|}.$$

Fig. 6d shows how to calculate $d(\mathbf{p}_k, l_e, \mathbf{n}_r)$; the centerness term $C(k, m) = e^{-\frac{(k-m/2)}{(m)^2}/2}$ is used to reduce the ambiguity of

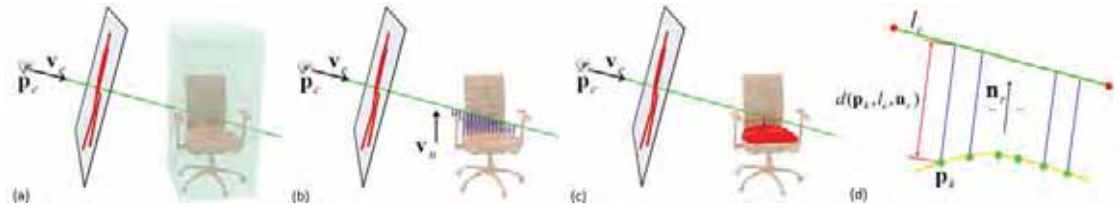

Fig. 6. Locating the sit position: (a) the ray from the eye to the pelvis node intersects the bounding box of the chair; (b) find all possible sitting areas on the chair; (c) determine the final sit position which is h (hip height) offset from the optimal sample point along the normal direction. (d) The optimal sample point is the one with the distance to the eye ray along the normal direction to be closest to h.

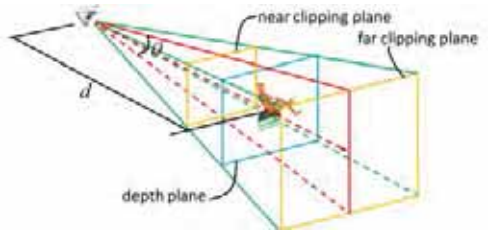

Fig. 7. Stick figure normalization.

the user input, and we assume that the user places the character at the center of the sitting area. The sitting region R_s is the region with the minimal metric across all regions (Fig. 6c), and the root position $\mathbf{p}_r = \mathbf{p}_{sk} + h \cdot \mathbf{n}_r$. In some cases, the hip may collide with the chair under the optimal sitting position. We then search those intersection points $(\mathbf{p}_{k-r} \dots \mathbf{p}_k \dots \mathbf{p}_{k+r})$ around the optimal sitting position for a certain range r ($r = 5$ in our current implementation) until we find a collision-free arrangement.

We also provide a scheme for users who prefer to draw the contact points instead of the center lines. Note such a style is also useful in some extreme cases which are difficult to specify the sitting position with center lines. For example, when the user wants to sit the character on a very thin object such as the arm of a chair. The user can choose between the two different sketching styles. For the contact-point method, we simply set $h = 0.0$ in (3) and rewrite (2) as $s_1 \le d \le s_2$ with $s_1 = -0.05$ and $s_2 = 0.05$.

4.1.3 Normalization

We also noticed that it is quite difficult for the user to estimate the appropriate projected bone length when drawing the stick figure. As shown in Fig. 5c, the length of some sticks are too long and out of the normal range. According to our observations, error rate could be 10-20 percent of the maximum bone length. Additionally, feedback from users indicated that more practice with our system would efficiently reduce such errors. Therefore, we checked and normalized the stick figure to eliminate such cases before pose reconstruction. Specifically, we determined the maximum bone projection length by assuming the bone lie on the depth plane of the succeeding joint (see Fig. 7)

$$\mathbb{P}l = \frac{h_v \cdot b}{2 \cdot d \cdot \tan\frac{\theta}{2}},$$

where h_v is the height of viewport, $d = d_r + \delta d$ is the distance from the eye position to the joint depth plane, θ is the field of view angle, and b is the bone length. Because the 3D pose is unknown, we cannot know the exact depth plane of each joint. Therefore, we take the depth plane of the sitting position for every bone, $d = d_r$.

We calculate the scaling factor s of the stick figure with the following equation:

$$s = min\left(\frac{\mathbb{P}l_i}{l_i}\right),$$

$s < 1.0$ means there exist some sticks whose length are greater than their normal range. We then scale the stick figure around the tree root by s.

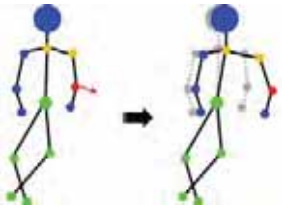

Fig. 8. Editing stick figure in global mode.

4.2 Stick Figure Editing

In our system, the user can edit the stick figure by dragging each stick joint. The editing operation could be either local or global. For the local case, only the dragging joint will be modified. While in the global mode, other joints may also be affected. By default, the system will automatically switch between the two editing modes. When editing operations happen during the drawing or when the dragging joint is d distance away from the original position ($d = 100$ pixels in our implementation), the editing stays in local mode. Otherwise, when the distance is greater than the threshold, the operation will automatically switch to the global mode. User can also choose to manually switch between the two editing modes. In the global mode, we first find those directly affected n joints which lie in between the dragging joint and the root joint, the orange ones in Fig. 8. These joints are updated by solving a linear system:

$$\mathbf{L} \cdot \mathbf{p} = \delta,$$

where $\mathbf{p}$ is an $n \times 2$ matrix, the two column vectors are the x and y components of joint coordinates. δ is also an $n \times 2$ matrix, consisting of the original Laplacian coordinates of the joints. The Laplacian coordinate of each joint is calculated by

$$\delta_i = \mathbf{p}_i - (\mathbf{p}_{i-1} + \mathbf{p}_{i+1})/2.$$

$\mathbf{L}$ is the Laplacian matrix, defined as

$$L_{ij} = \begin{cases} 1.0 & \text{if } i = j, \\ -0.5 & \text{if } i = j \pm 1, \\ 0.0 & \text{otherwise.} \end{cases}$$

The children of the affected joints and the dragging joints (blue dots in the stick figure) are moved rigidly following their parents. The other joints (green dots) in the stick figure keep unchanged. The purpose of the global editing mode is to propagate distortion around the dragging joint to the whole stick figure when it is edited significantly. The basic idea is to preserve the relative orientation between the neighboring sticks as much as possible. However, such a style only provides a convenient, but rough way, to edit the entire stick figure. Therefore, some editing results may be undesirable, which could change the stick lengths. In such cases, the user could further edit the undesired joints within the local editing mode.

4.3 Pose Reconstruction

4.3.1 Overview

The core of our character-posing system is the reconstruction of a 3D pose from a 2D stick figure. Previous works on pose reconstruction from a single image may not work well

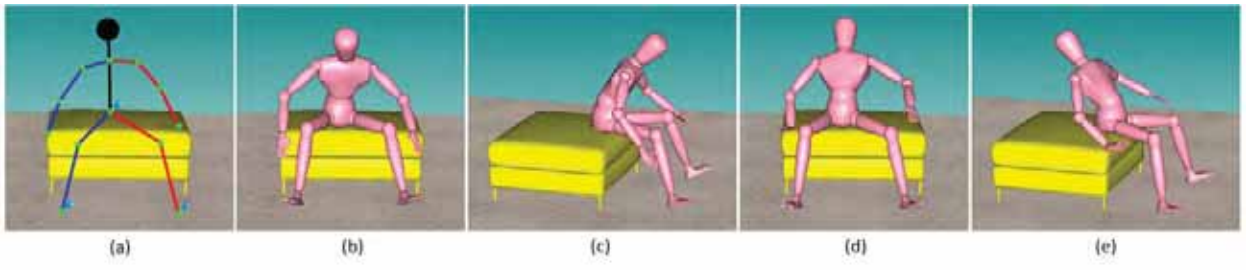

Fig. 9. Balance constraint: (a) 2D stick figure; (b, c) reconstructed 3D pose with balance constraint; (d, e) without balance constraint.

to the hand-drawn stick figure due to the imprecise nature of the stick figure.

We formulate the pose reconstruction problem as an optimization problem. The input of the reconstruction algorithm is the 2D positions of the joints in the stick figure ($\mathbf{s}_i$), and the output is the joint angles of the reconstructed 3D character pose ($\mathbf{q}_i$). We represent each joint angle with angle vectors with the dimension of the DOF of the joint. We use Euler angles to parameterize joint rotation as it is intuitive to place limits on the legal range of motion for them. Besides, the axis angle vector is a more appropriate parameterization for differential control with inverse kinematics [14]. The objective function evaluates the match between the projected 3D pose and the 2D input sketch, the plausibility of the pose, and the distance between the pinned joints and their bases. Collision handling is done separately. In the following, we first describe the details of the objective function and then describe how we solve this optimization problem. Finally, we describe how we handle collisions in this framework.

4.3.2 Objective Function

The objective function to determine the optimal 3D pose from a 2D sketch is described by the following equation:

$$E = w_p E_p(\mathbf{Q}) + w_b E_b(\mathbf{Q}) + w_a E_a(\mathbf{Q}), \tag{4}$$

where $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_n]$ is the vector of joint angles subject to the range of motion of each joint. $\mathbf{W} = [w_p, w_b, w_a]$ are the weights given to each term. In our implementation, we empirically set $w_p = 1.0$, $w_b = 0.5$, and $w_a = 5.0$ such that the priority of the energy terms (from high to low) is attach energy, projection energy, and balance energy.

The first term **projection energy** E_p measures the consistency between the projection of the reconstructed 3D pose and the 2D stick figure. We consider both the orientation and the position, with an emphasis on the orientation. Denoting the projection of the corresponding joint as $\mathbf{p}_i$, the orientation energy for sketch node $\mathbf{s}_i$ is defined as

$$E_{i\text{-}ori} = \sum_{j}^{c_i} \left(1 - \frac{(\mathbf{p}_j - \mathbf{p}_i) \cdot (\mathbf{s}_j - \mathbf{s}_i)}{\| \mathbf{p}_j - \mathbf{p}_i \| \cdot \| (\mathbf{s}_j - \mathbf{s}_i) \|}\right), \tag{5}$$

where c_i is the set of children nodes of the joint corresponding to sketch node $\mathbf{s}_i$. The position energy is described by the following equation:

$$E_{i\text{-}pos} = \frac{\| \mathbf{p}_i - \mathbf{s}_i \|^2}{\| \mathbf{s}_i - \mathbf{s}_r \|^2 + \sum \mathbb{P} l^2}, \tag{6}$$

where s_r is the coordinate of the root stick joint, $\sum \mathbb{P} l^2$ is the sum of the stick lengths in between the root stick joint and stick joint i. The projection energy is then defined as the sum of the energy of all sketch nodes

$$E_p = \sum_{i}^{n} (E_{i\text{-}ori} + w E_{i\text{-}pos}), \tag{7}$$

where w leverages the importance between orientation and position constraints. We empirically set $w = 0.1$ to emphasize more on the orientation, and thus to further reduce the noise contained in the user-drawn stick figure.

We introduce the second term, **balance energy** E_b, to keep the character balanced, leading to a physically plausible pose (Fig. 9). We achieve the balance constraint by forcing the center of mass of the whole body to stay over the supporting polygon [15] which is defined by the set of joints pinned to the environment. We minimize the distance between the ground projection of the character barycenter and the center of the supporting polygon

$$E_b = \| \mathbf{MC} - \mathbf{c} \|^2 . \tag{8}$$

$\mathbf{c}$ is the supporting center, matrix $\mathbf{M} = \mathbf{P} \cdot \mathbf{T}$ calculates the new barycenter of the character and projects it onto the ground, $\mathbf{T} = (m_1\mathbf{T}_1, m_2\mathbf{T}_2, \ldots, m_n\mathbf{T}_n)$ ($\mathbf{T}_k$s and m_ks are the rigid motion and the mass of each skeleton bone, respectively, $m_k s$ are normalized by the overall mass), $\mathbf{P}$ is the projection matrix, and $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_h]^T$ are the barycenters of the skeleton bones in standard pose. We currently only consider explicitly pinned joints in computing the supporting polygon. Little weight is assigned to this balance energy, so it is only used when the input sketch is very ambiguous, when it is helpful in eliminating some invalid poses (Fig. 9).

We name the third term **attach energy** E_a, which is used to constrain a joint attached to certain place (Fig. 10). There are two different attach styles: 1) the user specifies a joint attached to a limb (e.g., hand on knee); and 2) the user specifies a joint attached to the environment (e.g., feet on ground); If the selected sketch node $\mathbf{s}_k$ is very close to a nonneighbor sketch limb $\mathbf{s}_{k0}\mathbf{s}_{k1}$, we treat it as the first case (Fig. 10b). Otherwise, we generate an eye ray passing through the selected sketch node $\mathbf{s}_k$ in the screen space. Then, we find the intersection point $\mathbf{v}$ between the ray and the chair or the ground (Fig. 10c). For the first case, we minimize the distance between the joint $\mathbf{j}_k$ and the closest point on the bone segment $\mathbf{j}_{k0}\mathbf{j}_{k1}$

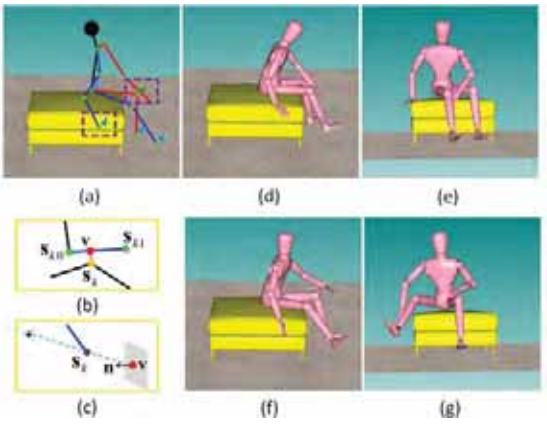

Fig. 10. Attach constraint: (a) stick figure; (b) attach style-1; (c) attach style-2; (d, e) with attached constraints; (f, g) without attached constraints.

$$E_a = \| \mathbf{j}_k - (\mathbf{j}_{k0} \cdot (1-t) + \mathbf{j}_{k1} \cdot t) \|^2$$
$$t = \frac{| \mathbf{s}_{k0}\mathbf{v} |}{| \mathbf{s}_{k0}\mathbf{s}_{k1} |}. \quad (9)$$

We define the energy for the second case with the following equation:

$$E_a = ((\mathbf{j}_k - \mathbf{v}) \cdot \mathbf{n})^2. \quad (10)$$

Here, we minimize the distance between the corresponding skeleton joint $\mathbf{j}_k$ and the plane defined by the attach point $\mathbf{v}$ and its normal $\mathbf{n}$ rather than the distance between the two points directly, as it is difficult to exactly specify the 3D position to which we want the joint to attach. Fig. 10 shows the difference between 3D figures generated with and without attach constraints.

4.3.3 Genetic Algorithm for Pose Reconstruction

We solved the above optimization problem using a two-phase approach. First, we solved this problem via Genetic Algorithm and second by refining the results with quasi-Newton method. The reason for this approach was twofold: 1) solving the problem with gradient methods, such as the quasi-Newton solver, requires an collision-free initial pose; it would be quite tedious for the user to construct a rough initial pose manually and also break our purpose in this paper to avoid direct manipulation of the 3D character and 2) genetic algorithm permits the investigation of multiple optima in the search space via niching techniques. Additionally, we argue that static poses, such as sitting poses, depend more on the character's intention, rather than styles or habits as in motion. Therefore, it would be better to provide multiple suggestive poses for user to reflect their intention.

Genetic algorithm is a heuristic search procedure that mimics the process of natural evolution to find the optimal solution. It has been applied to solve inverse kinematic problem [16]. However, there are few work on applying genetic algorithm for pose reconstruction [17]. Fig. 11 shows the flowchart of our GA solver for pose reconstruction:

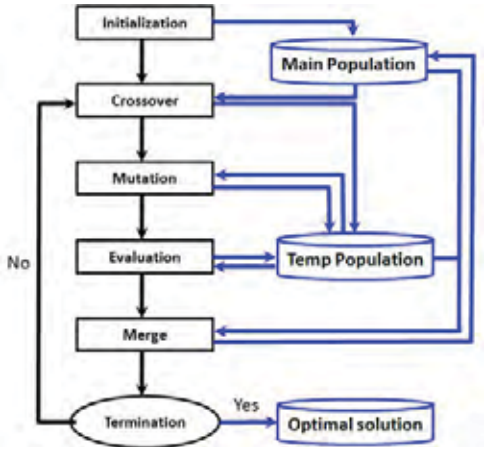

Fig. 11. Genetic algorithm for pose reconstruction.

- **Initialization**. An initial main population (with size S_m) of robot configurations is generated by random sampling of the variable search space. Each individual consists of n joint angles. The fitness values of the individuals are calculated by using (4). The population is sorted in increasing order of fitness value.
- **Crossover**. Crossover operation is applied onto the randomly chosen parent pair (P_1 and P_2) with a probability $P_c = 0.7$; otherwise, the two children (C_1 and C_2) will directly derived from the parents with the same property. The operation allows for rapid exploration of the search space. We use the simulated binary crossover (SBX) [18] to generate two children for the two parents. Among the n joint angles, l ($l = 0.5n$ in our implementation) of them are randomly selected for the crossover operation. The rest $n - l$ joint angles will be transferred from the parents to the children, remaining unchanged. For each of the selected l joint angles, a random number, $u_\beta(i)$, is generated. We then calculate a spread factor β using

$$\beta(i) = \begin{cases} (2u_\beta(i))^{\frac{1}{\eta+1}} & u_\beta(i) \le 0.5 \\ \left(\frac{1}{2(1-u_\beta(i))}\right)^{\frac{1}{\eta+1}} & otherwise. \end{cases}$$

And the children are produced from the following equation:

$$\begin{cases} C_1(i) = 0.5[(1+\beta(i))P_1(i) + (1-\beta(i))P_2(i)] \\ C_2(i) = 0.5[(1-\beta(i))P_1(i) + (1+\beta(i))P_2(i)]. \end{cases}$$

- **Mutation**. A further genetic operator, mutation is applied to the new individuals in temporary population. This operation makes local adjustments by perturbing m ($m = 0.1n$ in our implementation) of the n joint angles with a Gaussian term $N(0, \sigma^2)$.
- **Evaluation**. We then evaluate the fitness value for the temporary population generated through crossover and mutation. We also do collision detection for each individuals.
- **Merge**. We then incorporate the temporary population with population of current generation to form the next generation. We choose those collision free case in the temporary population by order of fitness value and replace those collided ones in the current generation.
- **Termination criterion**. After each evolution, we check the termination criterion. The genetic algorithm

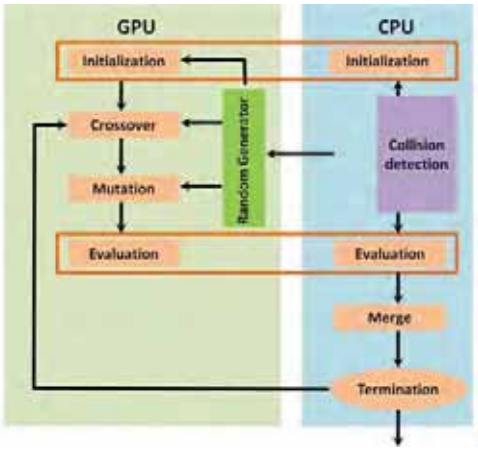

Fig. 12. GPU implementation of genetic algorithm.

will stop when one of three conditions is met: 1) the number of iterations reaches the maximum generation number; 2) The fitness value drops below the user-specified value (5.0); 3) The population has converged, i.e., the ratio of the population mean fitness divided by the maximum fitness dropped below the user-specified value (0.1).

4.3.4 GPU Implementation

A technical challenge in using GAs for optimization is that they are quite time consuming. A number of parallel genetic algorithms have been proposed running on parallel computers, and other specialized hardwares [19]. Several researchers have also implemented genetic algorithm on the GPU which is a much more convenient platform for general purpose parallel computation [20]. However, these implementations are designed for general problem and cannot be directly applied to our problem for several reasons: First, the fitness function in our application is complicated and depends on a lot of factors (scene transform matrix, stick figure, 3D skeleton, etc.). We need to design appropriate structure to store those information and to evaluate the fitness value efficiently. Second, the structure of our algorithm is different from their work which requires different organization. Finally, our algorithm involves a CPU-side collision detection procedure which also requires an appropriate organization. Fig. 12 shows the organization of our GPU-based GA solver. The crossover and mutation stage fully run on the GPU while the merge stage fully run on the CPU. For the initialization and the evaluation stage, we switch back to the CPU side for collision detection and run the other parts on the GPU.

Data organization. We need to pass the scene parameter, stick figure information and character skeleton information to the GPU. For the scene parameters, we pass the world-view-projection matrix and the viewport size. For the stick figure, we need to pass the screen coordinate of each stick joint. For the skeleton, we pass the relative position , relative orientation, and joint range of motion. Both the data for stick figure and skeleton are stored in a breadth-first manner. We store the main population and the temp population with a $S_m \times n$ and $S_t \times n$ array, respectively. We also store the updated absolute position and orientation of the skeleton joints for each genome in the main and temporary population. Those information will also be used to update the character mesh in collision detection except for fitness calculation.

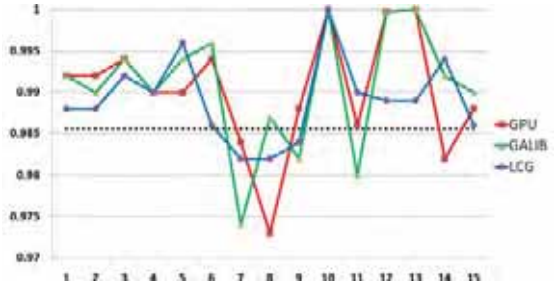

Fig. 13. Evaluation of the random number generator.

Random number generator. Random number generator plays an important role in the whole evolution process of genetic algorithm. We realize a pseudorandom number generators on the GPU for the genetic algorithm [21]. The GPU generator combines a Tausworthe generator with a linear congruential generator. We evaluate the randomness of the pseudorandom number generator using the NIST test suites [22]. The test suites contain 15 tests. The inputs for the tests are samples of binary sequences, and the outputs are p-values corresponding to the sequences indicating the probability that the result is due to random chance. We use a significance level of 0.01 with the sample size to be 500. Fig. 13 shows the test results. The vertical axis represents portion of passed sample while the horizontal axis represents the test list. The horizontal black dash lines mark the range of acceptable proportions which are determined by the significance level. For comparison purpose, we also test with several other popular algorithms provided either by the test suites (the Linear Congruential method) or by the GALIB. Notice that our purpose is just to show that our GPU generator is comparable to those algorithms.

4.3.5 Collision Handling

Collision handling is an important issue in our application. Therefore, we must ensure that there are no self-collisions on the polygonal mesh of the character or collisions between the character and the environment. Therefore, we took a more efficient strategy to handle collision in this paper. In previous work, we tried to avoid the generation of collision cases via trial-and-error; whenever a new genome was generated, we would check whether a collision occurred. If collision was detected, we would regenerate the genome until we yielded a collision free result. This process would repeat several times and if no collision-free genome was found, we would add the collided genome into the temp population. Although this method is quite effective, the scheme is also time consuming. In the current system, we treat the collision cases the same as those collision-free cases in the evolution operations. Further, we gradually replaced those collision cases in the main population with those in the temporary population.

Additionally, we omitted feet and hands in the stick figure to reduce the user's burden and kept the character feet and hands at standard 3D poses by default. We also ran

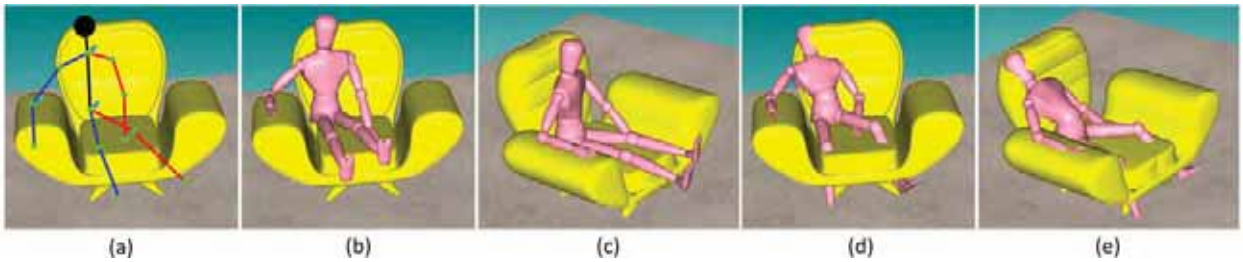

Fig. 14. Handling collision: (a) 2D stick figure; (b, c) reconstructed 3D pose with collision handling; (d, e) without collision handling.

collision detection for the feet and hands during the optimization process. Once a collision was detected, we search for collision-free orientations in the specific range of motion and assign them to the foot or hand in question.

The COLDET package [23] is used for collision detection. The collision handling is very helpful in eliminating unnatural poses, see Fig. 14 for an example.

4.3.6 Pose Refinement

Although genetic algorithms can effectively locate the region in which the global optimum exists, they take a relatively long time to locate the exact local optimum in the region of convergence. Therefore, the combination of a genetic algorithm and a local search method can improve the performance to locate the exact global optimum. We apply a quasi-Newton method to the solution generated by the genetic solver to further improve the result. The Jacobian matrix is computed numerically.

To handle the collision issue, we use a strategy similar to [24] by adding a collision energy to the objective function. If collision is detected at iteration k, we restore the angle vector to the previous collision-free iteration and then add new collision energy onto the objective function. For limb-limb collision, we find the two closest points $\mathbf{v}_1$ and $\mathbf{v}_2$ (Fig. 15a) and try to keep them as in collision-free iteration $k-1$. For limb-environment collision, $\mathbf{v}_2$ is replaced by the collision point in the environment (Fig. 15b)

$$E_{collision} = \left(\left\|\mathbf{v}_1^{[k]} - \mathbf{v}_2^{[k]}\right\| - \left\|\mathbf{v}_1^{[k-1]} - \mathbf{v}_2^{[k-1]}\right\|\right)^2. \quad (11)$$

In practice, this strategy is very effective to avoid most of the collisions. However, it cannot guarantee that the final result is completely collision free.

4.3.7 Attachment Enforcement

Although we introduced attach constraints in the objective function, the optimization can only place joints close to the attached positions, rather than accurately attach these joints the specified positions. Thus, we further conducted a rigid body dynamic simulation [25] on the reconstructed pose. For each skeleton chain that contain attaching joints, we performed a rigid body dynamic simulation with fixed root joint. The simulation ran until all attaching joints contacted the specified attach surfaces or until a certain time step number was reached. Here, we assumed that all the attach points would act as support points. Thus, this system cannot deal with these nonsupport attachment cases (e.g., attaching the hand to a wall, etc.). It also does not work well if the attachment locates on a very tiny object as in the case the attaching joint may fall outside the attaching region during simulation. In these cases, the user can manually drag the attaching joint to the target position.

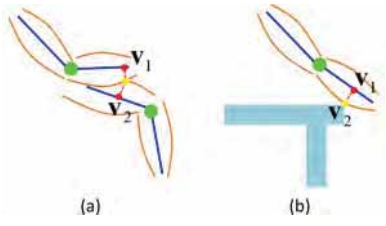

Fig. 15. Collision energy: (a) limb-limb collision; (b) limb-environment collision.

5 Results

We developed a prototype system in C++ on a workstation with an Intel Xeon 2.67-GHz CPU, NVIDIA Quadro FX 580 GPU. We conducted experiments in various virtual environments. See Figs. 1, 2, 9, 10, 14, and 16 for some examples of constructed 3D poses.

As shown in Fig. 17, a speedup of 3-5 can be obtained by using the proposed GPU solver, compared to the original CPU solver [2]. We also evaluated the reconstruction accuracy with respect to the population size. We run 50 reconstructions for each population size and computed the average fitness value and the standard deviation. As shown in Fig. 18, the bigger the population size, the smaller the fitness value and the standard deviation. Based on the above analysis, we set the default value of main population size to 1,024, which leads to accurate results at interactive speed (1-2 seconds to generate a pose). To investigate the effect of collision handling strategy on the convergence of the genetic solver, we conducted a test to reconstruct poses from the input in Fig. 2 with/without collision handling. As shown in Fig. 19, the collision handling strategy works pretty well as its convergence rate is very similar to the one without collision handling. We also compared the reconstructed result with the ground truth data. We first synthesized a 2D stick figure in the screen space from a posed character with 170 cm (see Fig. 20a). Then, we apply our algorithm on the synthesized stick figure to generate a reconstructed 3D pose (the pink one in Figs. 20b and 20c). The reconstruction error is 7.8 cm, which is computed as the average euclidean distances of 3D joints between the ground truth data and the reconstructed pose.

6 User Study

We conducted a user study to evaluate the benefits of our sketching interface for general user. Our system is

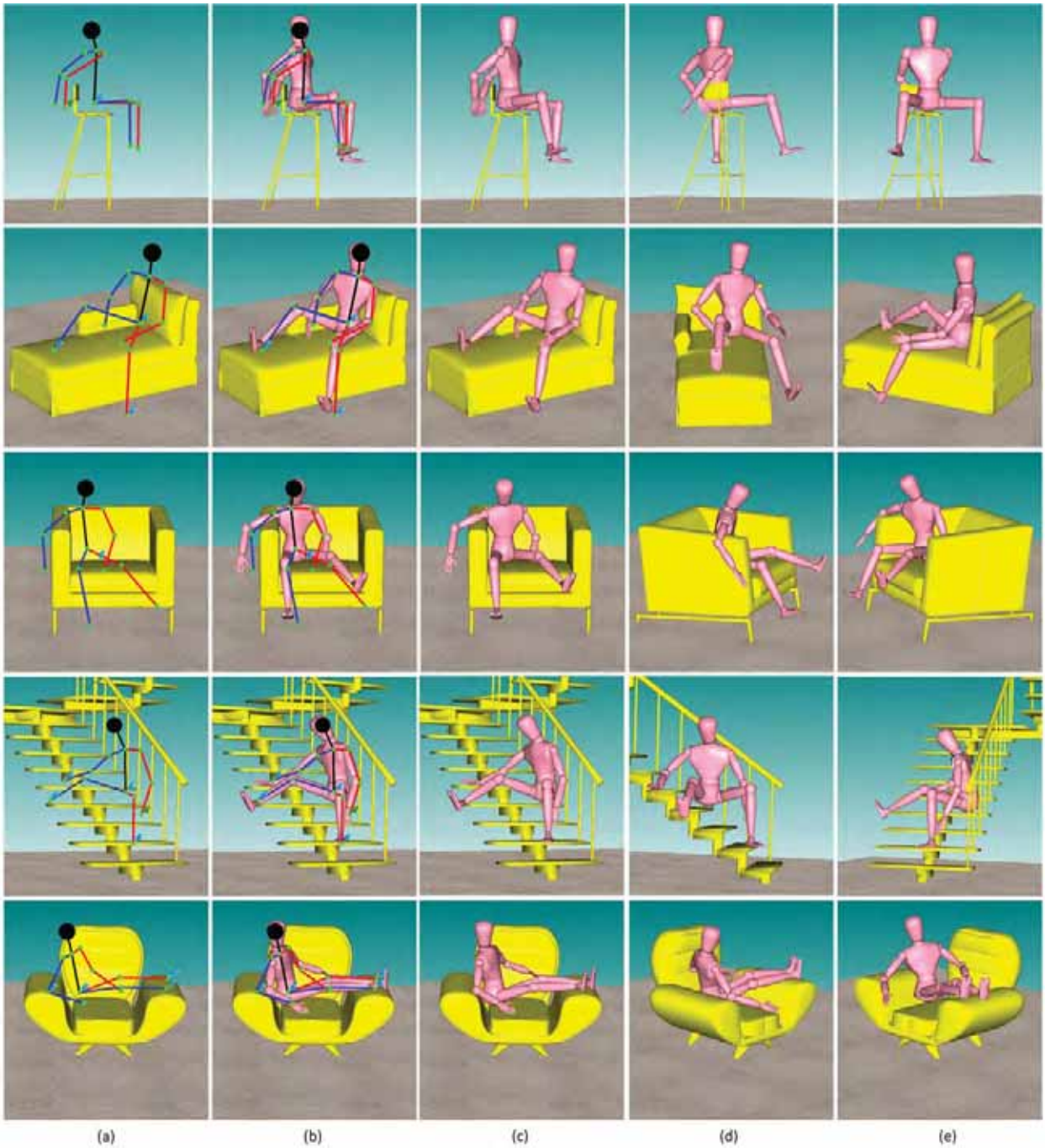

Fig. 16. More results. (a) The input stick figure; (b) overlay of the stick figure and reconstructed pose; (c)-(e) the reconstructed pose in different views.

compared against **Poser Pro 2010**, a professional 3D figure design and animation software. To pose a character with Poser Pro 2010, users can choose to switch on/off the inverse kinematics function. With inverse kinematics, users only need to edit the end joints, the intermediate ones will be updated automatically. Without inverse kinematics, users have to edit each joint one by one. We designed two experiments (with increasing difficulty) in the user study to evaluate how the interfaces can help user to design a 3D pose in a complex environment. The test data set of each experiment includes a photo showing the desired pose, the 3D virtual environment in both Poser Pro 2010 and our system (see Fig. 21).

Participants. Twenty participants were recruited in our user study: 12 males and eight females, aged between 20 and 35. Among them, four are professional artists, four have no experiences with 3D software, the others have passing knowledge or certain experiences in 3D design. Note that the familiarity with the tasks may affect the user study result. To minimize such a potential discrepancy, we divided the participants into two equally sized groups, the professional artists and beginners were split equally into the two groups. The first group (G1) was asked to complete the tasks using the proposed sketching interface followed by Poser Pro 2010, while the other group (G2) did exactly the same tasks but in the reverse order.

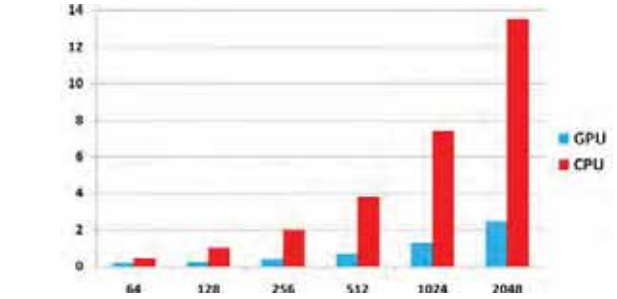

Fig. 17. Performance of the GPU solver and the original CPU solver. Horizontal axis: the population size; vertical axis: execution time (seconds).

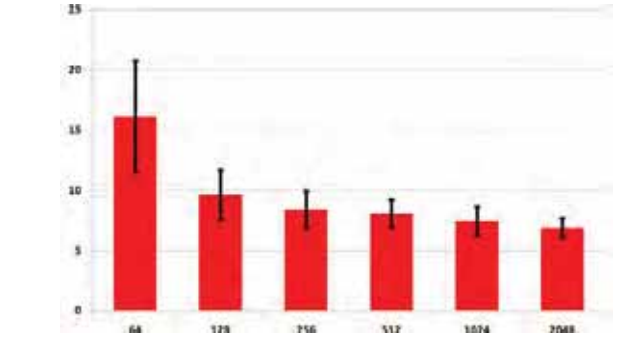

Fig. 18. Accuracy of the solver w.r.t. the population size. Horizontal axis: population size; vertical axis: fitness value.

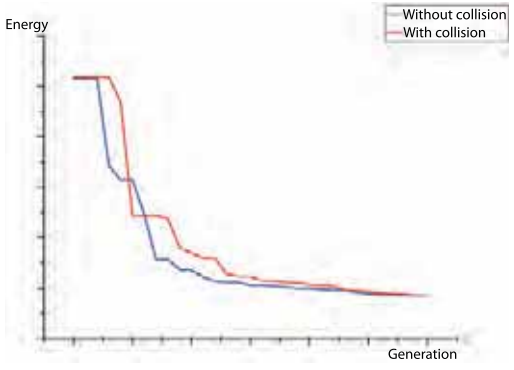

Fig. 19. Convergency of the genetic solver.

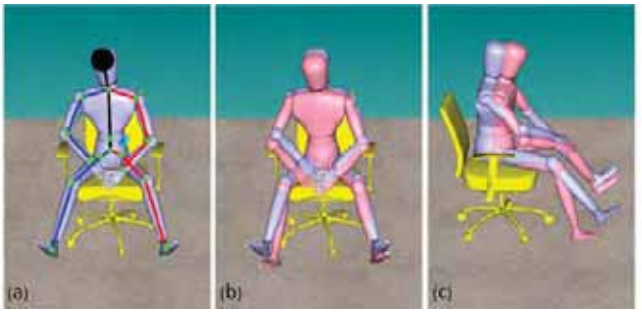

Fig. 20. Evaluation of reconstruction accuracy through comparison with ground truth data. (a) Ground truth data; (b, c) reconstructed pose and the ground truth data.

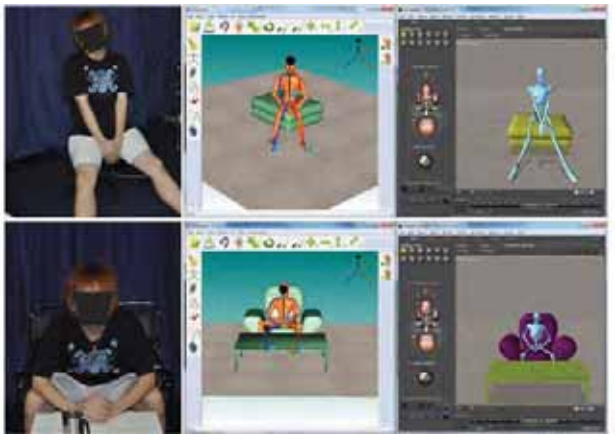

Fig. 21. Test data used in the user study. Left: the pose pictures shown to the user. Middle: the results of our sketching interface. Right: the results of Poser Pro 2010.

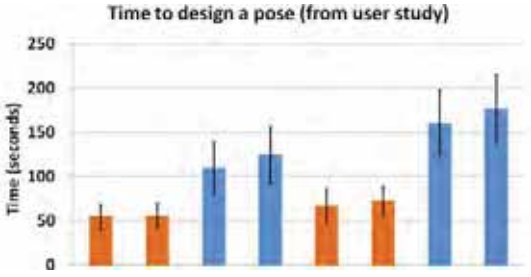

Fig. 22. User study results. We measured the time each participant took to complete the task in Experiments #1 and #2. G1 and G2 refer to the first and second groups, respectively, while S and P stand for our system (Sitter) and Poser Pro 2010, respectively. We show the average time and the 95 percent confidence interval for each case.

There are two stages in the user study. In the first stage (for G1), we first showed our proposed interface to the participants and gave them 5 minutes to try the operations in the interface. Then, they were asked to complete the two design tasks (two experiments) with it. After that, we moved on to the second stage and introduced the interface of Poser Pro 2010 to them. Again, they had 5 minutes to get familiar with the interface. After that, they were asked to do the (same) two experiments with Poser Pro 2010 this time. The second group did exactly the same tasks but in a reverse order when using the two systems. During the course of the user study, the time taken by each participant to complete a task was measured.

Experiment #1. The first experiment is a character sitting on a stool and stretching his legs. It aims to evaluate how the 2D sketching interface can help users to quickly design a 3D pose. Here, we choose to use a simple pose with simple environment, so as to eliminate the interferences of other factors such as occlusion, collision, etc. With Pose Pro 2010, the users had to frequently change the viewpoint to find the appropriate viewing direction and specify the joint locations. In sharp contrast, our sketching interface allows the users to simply sketch a stick figure without changing the view direction, pin the feet on the ground and then obtain the 3D pose immediately. As shown in Fig. 22, the participants saved up to 53.4 percent time by using our sketching interface. We conduct the paired t-test to investigate the significance of the difference between the averaged time costs of the two interface. We have $t = 5.7771$, $df = 19$ and $p < 0.0005$; thus, we conclude that our result is significant beyond the 0.0005 level.

Experiment #2. The second experiment is slightly difficult than the first one in that the legs are occluded by a tea table placed in front of the character. This experiment

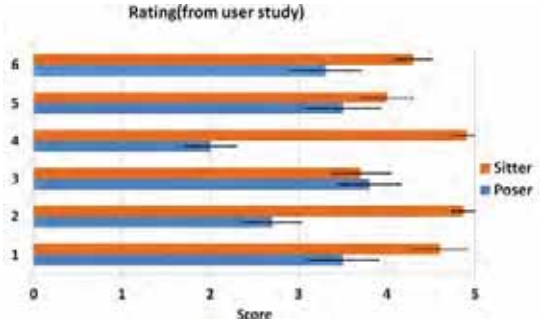

Fig. 23. Rating of the two testing systems (Sitter: our interface, Poser: Poser Pro 2010) on a scale between 1 and 5 ($1 =$ strongly disagree, $5 =$ strongly agree), the error bars represent for the 95 percent confidence interval.

aims to simulate a complex environment, which is quite common in real-world design applications. Thus, the pose and the environment shown to the user are much complex than the previous one, frequent switch of camera will be involved. Due to the occlusions, the Pose Pro 2010 users have to carefully align the camera to position the joints; thus, it is very tedious to position the legs (see the supplementary video, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.61). Besides, using a 2D input device such as mouse for such kind of task is also a challenge. The consideration of collision with the environment makes the task even more difficult. The users of our system, however, can complete this task in the same way as the first experiment, since they can sketch the 2D stick figure on the screen space without caring the occlusion issue at all. The time statistics in Fig. 22 justify that our sketching interface is more efficient than the conventional IK-based interface saving 58.7 percent time. According to the paired t-test ($t = 8.9183$, $df = 19$, $p < 0.0005$), the result is significant beyond 0.0005 level.

Note that the sitting poses and the environment configurations in the two experiments are carefully designed for the comparison of the two interfaces. They maybe not sufficient to cover the whole range of sitting pose; however, they are typical and very common in real life.

Rating. After the two tasks, the participants were also asked to rate the two interfaces on their "preference," "effectiveness," "accuracy," and "easy to use" on a scale of 1 to 5: 1 means "strongly disagree" and 5 means "strongly agree." Our interface received very positive feedbacks as shown in Fig. 23. Some participants also commented that Pose Pro 2010 can lead to more accurate result than our interface, since one can directly position each joint, although this manipulation is very tedious. 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 1 shows the test result.

TABLE 1
Wilcoxon Signed Rank Test Result

Question	1	2	3	4	5	6
Z value	1.952	2.790	-0.209	2.772	1.643	2.809
P value	0.025	0.003	0.417	0.003	0.050	0.002

Fig. 24. Failed case: pose that cannot be generated with our system.

7 Limitations

Our current implementation has several limitations. First, we considered only simple collision and balance constraints in our current implementation. In fact, the interaction between the character and the environment could be more complicated. Second, the determination of supporting polygon completely depends on how the user specifies with the pinning tool. It is desired to automatically detect some extra contact points to further reduce the user's burden. Third, our system does not work for some highly complicated poses such as the Lotus position in Yoga, where both feet are placed in front of the pelvis with knees bent (see Fig. 24).

8 Conclusion

This paper presented an intuitive sketching interface for a sitting pose design in a virtual environment. Our interface only requires the users to draw a simple 2D stick figure on the screen space without changing the viewing direction. Following this sketch, physically correct and visually pleasing 3D poses are reconstructed, automatically, at an interactive speed. Our system is novel in that it takes into account the interaction between the character and environment, which is helpful in solving the ill-posed 2D-to-3D reconstruction problem. The promising experimental results and the user study demonstrate that our method is efficient, intuitive, and effective and allows the users to quickly design the desired sitting pose in a complex virtual environment.

Of note, our current framework focuses only on a sitting pose design. However, it can be easily extended for other static pose design tasks such as lying or standing as shown in Fig. 25. We can also choose to specify other joints as the main pivot by making the joint the root of the skeleton. Finally, the technique described in [26] could be used for handling such a structure-varying skeleton.

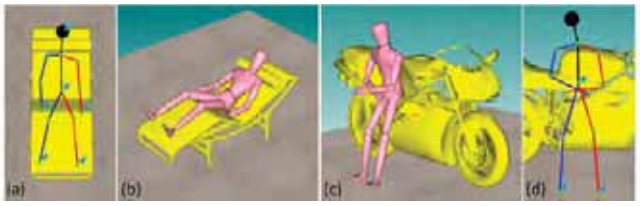

Fig. 25. Posing character in other environments.

ACKNOWLEDGMENTS

This project was partially supported by NRF2008IDM-IDM004-006. The authors would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] G. Saul, M. Lau, J. Mitani, and T. Igarashi, "Sketchchair: An All-In-One Chair Design System for End-Users," *Proc. Fifth Int'l Conf. Tangible, Embedded and Embodied Interaction,* 2011.
[2] J. Lin, T. Igarashi, J. Mitani, and G. Saul, "A Sketching Interface for Sitting-Pose Design," *Proc. Eurographics Symp. Sketch-Based Interfaces and Modeling,* pp. 111-118, 2010.
[3] J. Zhao and N. Badler, "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures," *ACM Trans. Graphics,* vol. 13, no. 4, pp. 313-336, 1994.
[4] K. Grochow, S.L. Martin, A. Hertzmann, and Z. Popovic, "Style-Based Inverse Kinematics," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 522-531, 2004.
[5] J. Chai and J.K. Hodgins, "Performance Animation from Low-Dimensional Control Signals," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 686-696, 2005.
[6] J. Davis, M. Agrawala, E. Chuang, Z. Popovic, and D. Salesin, "A Sketching Interface for Articulated Figure Animation," *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation,* pp. 320-328, 2003.
[7] X. Wei and J. Chai, "Intuitive Interactive Human-Character Posing with Millions of Example Poses," *IEEE Computer Graphics and Applications,* vol. 31, no. 4, pp. 78-88, July/Aug. 2011.
[8] E. Jain, Y. Sheikh, and J.K. Hodgins, "Leveraging the Talent of Hand Animators to Create Three-Dimensional Animation," *Proc. ACM SIGGRAPH/EUROGRAPHICS Symp. Computer Animation,* pp. 93-102, 2009.
[9] W. Yoshizaki, Y. Sugiura, A.C. Chiou, S. Hashimoto, H. Inami, T. Igarashi, Y. Akazawa, K. Kawauchi, S. Kagami, and M. Mochimaru, "An Actuated Physical Puppet as an Input Device for Controlling Digital Manikin," *Proc. ACM CHI Conf. Human Factors in Computing Systems,* 2011.
[10] C. Barron and I.A. Kakadiaris, "Estimating Anthropometry and Pose from a Single Image," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 669-676, 2000.
[11] C. Taylor, "Reconstruction of Articulated Objects from Point Correspondences in a Single Uncalibrated Image," *Computer Vision and Image Understanding,* vol. 80, no. 3, pp. 349-363, 2000.
[12] X.K. Wei and J. Chai, "Modeling 3D Human Poses from Uncalibrated Monocular Images," *Proc. IEEE Conf. Computer Vision,* pp. 1873-1880, 2009.
[13] A. Agarwal and B. Triggs, "Recovering 3D Human Pose from Monocular Images," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 28, no. 1, pp. 44-58, Jan. 2006.
[14] F.S. Grassia, "Practical Parameterization of Rotations Using the Exponential Map," *J. Graphics Tools,* vol. 3, no. 3, pp. 29-48, 1998.
[15] C.B. Phillips and N.I. Badler, "Interactive Behaviors for Bipedal Articulated Figures," *Computer Graphics,* vol. 25, no. 4, pp. 359-362, 1991.
[16] S. Tabandeh, W.W. Melek, and C.M. Clark, "An Adaptive Niching Genetic Algorithm Approach for Generating Multiple Solutions of Serial Manipulator Inverse Kinematics with Applications to Modular Robots," *J. Robotica,* vol. 28, pp. 493-507, 2010.
[17] J. Zhao, L. Li, and K.C. Keong, "3D Posture Reconstruction and Human Animation from 2D Feature Points," *Computer Graphics Forum,* vol. 24, no. 4, pp. 759-771, 2005.
[18] K. Deb and R. Agrawal, "Simulated Binary Crossover for Continuous Search Space," *Complex System,* vol. 9, no. 2, pp. 115-148, 1995.
[19] M. Tomassini, "A Survey of Parallel Genetic Algorithms," *Annual Reviews of Computational Physics,* pp. 87-118, World Scientific, 1995.
[20] M.-L. Wong and T.-T. Wong, "Parallel Hybrid Genetic Algorithms on Consumer-Level Graphics Hardware," *IEEE Congress Evolutionary Computation,* pp. 2973-2980, 2006.
[21] H. Nguyen, *GPU Gems 3.* Addison-Wesley Professional, 2007.
[22] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," technical report, Nat'l Inst. of Standards and Technology, 2010.
[23] COLDET, "Free 3D Collision Detection Library," http://sourceforge.net/projects/coldet, 2012.
[24] X. Shi, K. Zhou, Y. Tong, H. Bao, and B. Guo, "Mesh Puppetry: Cascading Optimization of Mesh Deformation with Inverse Kinematics," *ACM Trans. Graphics,* vol. 26, no. 3, article 81, 2007.
[25] R. Smith, "Ode: Open Dynamic Engine," http://www.ode.org/, 2010.
[26] Y. Nakamura and K. Yamane, "Dynamics Computation of Structure-Varying Kinematic Chains and Its Application to Human Figures," *IEEE Trans. Robotics and Automation,* vol. 16, no. 2, pp. 124-134, Apr. 2000.

Juncong Lin received the PhD degree in computer science from Zhejiang University in 2008. He is an associate professor in the Software School of Xiamen University, China. His research interests include digital geometry processing and sketch-based modeling.

Takeo Igarashi received the PhD degree from the Department of Information Engineering at The University of Tokyo in 2000. He is a professor in the Department of Computer Science, The University of Tokyo and director of Igarashi Design Interface Project, JST/ERATO. His research interest is in user interfaces and interactive computer graphics. He has received ACM SIGGRAPH Significant New Researcher Award in 2006.

Jun Mitani received the PhD degree in precision engineering from the University of Tokyo. He is an associate professor of computer science at the University of Tsukuba. His research interests include computer graphics, geometric modeling, and origami.

Minghong Liao received the PhD degree in computer science from Harbin Institute of Technology of China in 1993. He is a professor of Software School of Xiamen University. His research interests include Intelligent Network, Pervasive Computing and Computer Graphics.

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 currently an assistant professor at the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests fall in the broad area of visual computing. He is particularly interested in the problems that require geometric computation and analysis. He is a member of the IEEE. More information about his research can be found at http://www.ntu.edu.sg/home/yhe.