Implementation of Metalworking Mode in Mixed Reality Modeling System Using ToolDevice

Ryan Arisandi, Mai Otsuki, Asako Kimura, Fumihisa Shibata, and Hideyuki Tamura

Ritsumeikan University, Shiga, Japan
(Tel: +81-77-566-1111, E-mail: arisandi@rm.is.ritsumei.ac.jp)

Abstract: *ToolDevice is a set of interaction devices developed to help users in spatial work such as layout designing and three-dimensional (3D) modeling. ToolDevice consists of three components: TweezersDevice, Knife/HammerDevice, and BrushDevice, which use metaphors of real-life hand tools to help users recognize each device's unique functions. For TweezersDevice and Knife/HammerDevice, we have developed a mixed reality (MR) 3D modeling system that imitates real-life woodworking. In the system, TweezersDevice is used to pick up and move, while Knife/HammerDevice is used to cut and join virtual objects represented as wood materials. Unfortunately, the system lacks a common function for manipulating the shape of virtual objects. In this paper, we propose a novel way for manipulating the shape of virtual objects using ToolDevice. We also present the results of an informal user study conducted to verify the intuitiveness of our proposed method. Finally, we discuss the problems users reported in the study and future work.*

Keywords: Interaction Device, Shape Manipulation, Tool Metaphor

1. Introduction

Most three-dimensional (3D) modeling software is not made for beginners. The user interface is often complex and requires users to have knowledge of mathematics. Studies have proposed new approaches that make the 3D modeling process more user friendly [1]. One approach is to use real-life parts connected to the computer [2]. Another is to use body parts, such as fingers, to manipulate virtual objects [3][4]. The most common approach is to use devices made specifically for the assumed operations [5][6]. However, these devices are often abstract in form, confusing novice users on how to use them.

We have developed ToolDevice, a tangible user interface that uses metaphors of real-life hand tools [7]-[9] (Fig. 1). ToolDevice is a set of interaction devices that offers two key features:

It employs the familiar shapes and tactile sensations of hand tools that are commonly used in everyday life.

It is based on the idea that different tools are used for different purposes.

These features help novice users recognize the use and functions of devices by just looking at them. In addition, users can apply these devices for digital operations in a similar manner to real-life operations.

Our focus is to help users who have difficulty in performing spatial work, such as layout designing and 3D modeling, by using a traditional two-dimensional (2D) display and mouse. For this reason, our devices are used in an immersive 3D environment. For simplicity, we generalized the operations of spatial work into three types: picking up and moving, manipulating, and painting. We developed one device for each type of operation:

(1) TweezersDevice imitates the shape of a pair of tweezers for picking up and moving virtual objects.

(2) Knife/HammerDevice is specially designed to change the tip to that of a knife or a hammer, depending on the need. The knife tip is designed for cutting virtual objects, whereas the hammer tip is designed for joining virtual objects. In either case, the user can hold the device just like holding a real-life knife or hammer.

(3) BrushDevice that imitates the shape of a brush and is designed for painting virtual objects.

For TweezersDevice and Knife/HammerDevice, we have developed a mixed reality (MR) 3D modeling system that imitates woodworking in real life [9]. In this system, users can see virtual wood materials through a head-mounted display (HMD), pick up and move them using TweezersDevice, and cut objects using KnifeDevice. By changing the knife tip to hammer tip and swinging the device onto adjoining virtual objects, users can join multiple objects into one. The operation resembles hitting a nail with a hammer to join several components. To keep the operation simple, we require users to hit virtual objects only once to join them. Unfortunately, our system lacks one fundamental function, shape manipulation. Without this function, users cannot create a complex-shaped object, such as the letter S. Although users can cut bigger objects into smaller pieces and rejoin them to create complex-shaped objects, this operation is tedious and time consuming.

Therefore, the goal of this research is to implement an intuitive shape manipulation function into our modeling system using ToolDevice. However, because shapes are not manipulated using hand tools in real-life woodworking, we expect our users to have difficulties

Fig. 1 ToolDevice: (a) TweezersDevice, (b) Knife/HammerDevice, and (c) BrushDevice

in associating operations in the system with real-life operations. We solve this problem by splitting our modeling system into two modes: woodworking mode and metalworking mode. We will explain this in more detail in section 3 after reviewing related work in section 2.

The rest of the paper is structured as follows. We explain our algorithm for implementing the shape manipulation function in section 4 and report the results of an informal user study that we conducted to confirm the intuitiveness of our proposed method in section 5. Finally we discuss the future direction of this work in section 6.

2. Related Work

Traditional 3D modeling software often requires users to have previous knowledge of mathematics in order to be able to manipulate the shape of virtual objects. This makes it difficult for novice users to learn how to use the software.

To solve this problem, a variety of approaches have been suggested for manipulating the shape of virtual objects. The challenge is to propose a method that is intuitive, can be easily controlled, and is applicable in a real-time environment.

Grossman *et al*. [10] developed TapeWidget, a high degree-of-freedom (DOF) input device made of rubber with a flexible spring steel core. Using the TapeWidget, users can create and edit virtual objects by performing gestures such as cracking, stubbing, and twisting. Llamas *et al*. [11] proposed a method of manipulating virtual objects using a pair of rigid handles with three buttons each. The handles can be used simultaneously to change the shape of virtual objects. McDonnell *et al*. [4] developed Virtual Clay, a system that enables users to manipulate the shape of virtual objects using a finger. The finger is tracked by a haptic device called PHANToM. Similarly, Sheng *et al*. [3] developed an interface that enables users to directly manipulate the shape of virtual objects using multiple fingers. The fingers are tracked using a Vicon motion capture system, and the virtual object is represented in the real world by a deformable physical prop. The prop acts as a proxy to the virtual object; by performing gestures on the prop, users can select, move, rotate, and manipulate the shape of virtual objects.

Although the works mentioned above are arguably more intuitive than traditional 3D modeling software in that they provide different ways for users to directly manipulate virtual objects, they still use traditional 2D displays. This limits the users' view to a planar surface. Consequently, operations often do not go as expected, especially operations involving parts that users cannot see.

Huang *et al*. [2] solved this problem by developing Easigami, a tool that enables users to create and manipulate virtual objects using a paper-folding metaphor. Easigami has a set of flat polygons of different shapes with embedded PCI microcontrollers that are used to connect to a computer. Users can connect the polygons with each other and fold them to create a model. This model will then be recreated in the computer. By using this technique, Easigami enables users to have a full 360° view of the model in the real world. However, the models' shapes and sizes are limited to those of the available polygons.

Choi *et al*. [12] developed a sketching system in an immersive 3D environment. Users can create and manipulate the shape of virtual objects using a wand-shaped wireless 3D input device. The wand is used as if the user were making a sketch in 3D space. This system enables users to freely create and manipulate 3D models without any restrictions. However, the wand-like input device makes it difficult for novice users with no previous experience to understand how to operate the device, especially for manipulating shapes.

Wesche *et al.* [5] developed FreeDrawer, an immersive sketch-based 3D modeling system similar to that in [12]. However, users must first open a tool selection menu and change the drawing tool before manipulating the shape of virtual objects. Furthermore, the authors concede that their system is not suitable for beginners.

As explained in the previous section, we have developed an MR 3D modeling system that enables users to cut and assemble virtual wood materials using ToolDevice. Adopting an MR environment allows users to not only view virtual objects without any restrictions but also freely place the objects wherever they like. Moreover, ToolDevice enables users to manipulate virtual objects by performing gestures that closely resemble real-life operations, such as picking up and moving (TweezersDevice), cutting (KnifeDevice), and joining (HammerDevice).

We realize that a shape manipulation function is necessary. However, because shape manipulation using hand tools is uncommon in woodworking, we assume that users will have difficulty in associating any type of operations in the system with real-life operations. To solve this problem, we examined how to best implement a shape manipulation function.

3. Design

3.1 Shape Manipulation Function

We examined what type of operations and tools are typically used in real-life handcrafting. Aside from woodworking, which was used as a metaphor in our previous work, we focused on three types of handcrafting that are most common: metalworking, clay modeling, and stone masonry.

We found that shape manipulation is most commonly done in metalworking and clay modeling. Metalworking uses hand tools such as a hammer to bend or dent metal materials. In contrast, in clay modeling, clays are manipulated by pressing or extruding them using hands. As we mentioned previously, our goal is to implement a shape manipulation function using ToolDevice. Therefore, we decided to adopt metalworking as a metaphor and implement a shape manipulation function

using HammerDevice. We assume that users can easily understand the operations in the system because they resemble real-life operations.

To avoid confusion with the already implemented joining function, which also requires HammerDevice, we split our modeling system into two modes: woodworking mode, in which users can assemble multiple virtual objects, and metalworking mode, in which users can manipulate the shape of objects (Fig. 2). Note that other functions such as moving and cutting are also available in metalworking mode via TweezersDevice and KnifeDevice, respectively.

3.2 Real-life Metalworking

We further examined what types of shape manipulation operations are usually done in real-life metalworking. We found that there are three common operations: denting, bending, and twisting. Of these three, we focused on denting and bending because both these operations are done using a hammer.

We decided to implement both denting and bending to give users more flexibility when manipulating the shape of virtual objects. However, because both the operations use HammerDevice, we must implement the operations in such a way that users can easily distinguish when a virtual object must be dented or bent.

3.3 Operations

When making a dent in metal materials, users first determine the part that is to be dented and place it on a platform. Users then hit the metal material with a hammer until the desired dent is achieved.

When bending metal materials, users first fix the material on a platform, with the part that is to be bent extending beyond the platform. Users then hit the protruding part with a hammer until the desired bend is achieved. When users hit a material, no reaction force occurs, and the material is bent (Fig. 3b).

For denting metal materials, users fix the material on a platform, placing the part that is to be dented anywhere on the platform, except on the edge. When users hit a metal material, a reaction force occurs, and the material is dented (Fig. 3a). In other words, the presence of a reaction force determines whether a metal material is dented or bent when hit by a hammer.

On the basis of this difference, we propose the following method of operation:

When denting a virtual object, users can place the object anywhere except on the edge of the real table used in the system. Hitting the object with HammerDevice will then create a depression on the object.

When bending a virtual object, users must place the object on the edge of the real table used in the system with the part that is to be bent extending beyond the table. By hitting the protruding part iteratively with HammerDevice, users can bend the object as much as they like.

We argue that when bending virtual objects, users need to be able to distinguish the part that is to be bent and that is to remain the same. For this reason, we require users to place virtual objects on the edge of the real table used in our system. In contrast, when making a dent in virtual objects, users need to determine only the part that they want to dent.

In addition, we assume that placing virtual objects on some type of platform every time users want to make a dent is bothersome and limits flexibility. Therefore, we made it possible for users to make a dent in virtual objects anywhere except on the edge of the real table.

3.4 Parameters

To realize our proposed method of operation, we analyzed the parameters that are considered when users dent and bend metal materials in real life. Each parameter for each operation is explained in detail below. However, these parameters are not used for rigorous simulation of shape changes because that is not our aim. These parameters are used to realize an intuitive method of manipulating the shape of virtual objects.

3.4.1 Denting

Three parameters are considered, as explained below.

(1) Hitting point: Naturally, when users want to make dent in a metal material, they will first determine the part of the material they want to dent. In most cases, users will hit the center point of the chosen part.

(2) Hitting power: After the denting point has been determined, the user will then proceed to hit the metal material with a hammer until the desired dent is achieved. The swinging speed of the hammer affects the hitting power, which in turn affects the size and depth of the dent. The harder the metal material is hit, the bigger and deeper the dent will be.

(3) Hitting direction: Depending on how the user wants the metal material to be dented, the material can be hit from any direction. For example, the dent will differ in shape depending on whether the user hits the material perpendicularly or diagonally.

3.4.2 Bending

Two parameters are considered, as explained below.

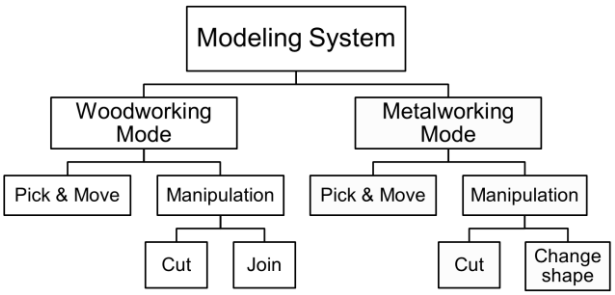

Fig. 2 Structure of modeling system

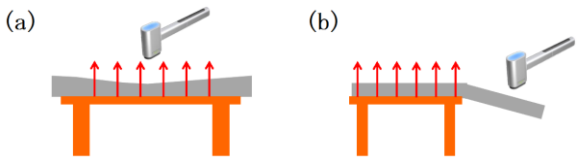

Fig. 3 Reaction force: (a) Denting (b) Bending

(1) Object placement: When bending a metal material, users will first determine how they want the material to be bent. Users will then fix the material on a platform with the part that is to be bent extending beyond the platform.

(2) Hitting power: After the metal material has been fixed on a platform, the user proceeds by hitting the protruding part. However, unlike denting, in this case, the hitting power affects only how far the material is bent.

4. Implementation

4.1 System Configuration

The system configuration is shown in Fig. 4. The specifications for the main PC are as follows: Microsoft Windows XP OS, Intel Core i7 Ext 965 CPU, and 6144 MB of RAM. We also use a binocular see-through HMD (Canon VH-2002) that enables users to perceive depth. The HMD is connected to a video capture card (ViewCast Osprey-440) that captures input videos from the cameras built into the HMD. The NVIDIA GeForce GTX 280 graphics processor is used for image processing. The positions and orientations of the HMD and ToolDevice are tracked using Polhemus LIBERTY, a six-DOF tracking system that uses magnetic sensors. A transmitter is also used as a reference point for the sensors. We use two devices from ToolDevice, TweezersDevice, and Knife/HammerDevice. The devices are connected to the main PC through an input/output (I/O) box. The I/O box retrieves information from the devices and sends it to the main PC, which then sends back commands to control the devices.

All the code in the system is written in C++/CLI in .NET Framework. We used OpenGL and the OpenGL Utility Toolkit (GLUT) for the graphics API. In creating the MR space, first we set the videos captured by the Osprey-440 as the background, and then we created a virtual viewing point in OpenGL by obtaining the position and orientation of the HMD from Polhemus LIBERTY. By doing this, we were able to make users feel as if they are manipulating virtual objects in the real world.

4.2 Shape Manipulation

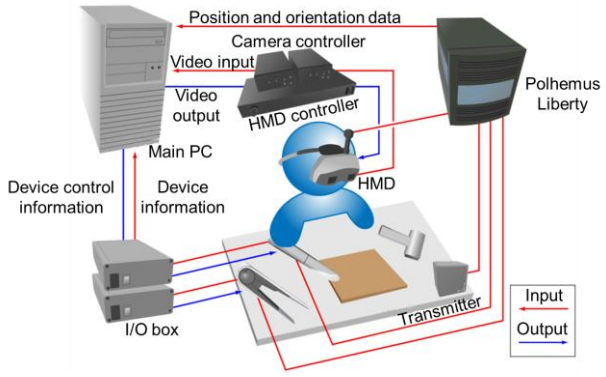

Fig. 4 System configuration

4.2.1 Object Triangulation

Our previous modeling system uses three types of data to represent a virtual object: vertices, edges, and surfaces. For example, a simple cube is defined by 8 vertices, 12 edges, and 6 surfaces. The advantage of this technique is that it requires low computational effort. However, it cannot represent objects with complex shapes, especially objects that contain many curves, such as the letter S. Therefore, we need to divide the surfaces of the virtual objects into smaller triangles.

Several methods can be used to divide surfaces into smaller triangles, also called triangulation; the most common are the ear-clipping method, the monotone polygon method, and Delaunay triangulation. The ear-clipping and monotone polygon methods use existing vertices to divide surfaces. In contrast, Delaunay triangulation allows users to determine various conditions in advance, such as how small the divisions of the surface should be, and add new vertices and edges to meet the previously set conditions. For better representations of complex-shaped objects, we decided to divide the surfaces of virtual objects using Delaunay triangulation.

To do this, we used a program called "Triangle" developed by Jonathan Shewchuk from the University of California [13]. Triangle allows users to easily set conditions by inputting command lines. In addition, it uses the same data structure as our system to represent virtual objects. Triangle stores vertex data in ".node" files, edge data in ".edge" files, undivided surface data in ".poly" files, and divided surface data in ".ele" files.

However, Triangle is restricted to triangulating 2D surfaces. Because our virtual objects are 3D, we need to convert the surfaces of our virtual objects into 2D surfaces and back to 3D surfaces several times. The steps to do this are explained in detail below; we will refer to this process as step 0.

(0a) 3D to 2D: For each surface, create multiple .poly files to store the surface's data. To convert the surfaces into 2D surfaces, follow the below steps:

(0a-1) Calculate the normal vector for each surface.

(0a-2) From the X, Y, Z values of the normal vector, determine the largest value, and write the two other values to the corresponding .poly file. For example, if the largest value is X, write only the Y and Z values.

(0a-3) Store the data for one 3D vertex from each surface to be used as a reference point.

(0b) 2D to 3D: For each surface, read the data from the .node files created in step (1-2) to store each surface's vertices. This step will be explained later. To convert the surfaces into 3D surfaces,

(0b-1) Calculate the D value of each surface. D is a variable in the plane equation

$$Ax + By + Cz + D = 0, \quad (1)$$

where A, B, C are the X, Y, Z values of the surface normal vector, and x, y, z are the X, Y, Z values of the reference point stored in step (0a-3).

(0b-2) After the D value has been determined, use Eq. (1) to convert the 2D vertices back to 3D by calculating the missing x, y, or z value of each vertex.

Triangulation itself requires several steps, as explained below. Triangulation is illustrated in Fig. 5.

(1) Add new vertices using Triangle.

(1-1) Convert virtual object's surfaces to 2D using step (0a).

(1-2) Set how small the surfaces should be divided and execute Triangle to divide the surfaces. The vertices data for the newly divided surface is then stored in the corresponding .node files.

(1-3) Convert the newly added vertices back to 3D using step (0b).

The result of this step is shown in Fig. 5b.

(2) Add vertices that are located at the intersection of surfaces to the adjoining surface. To achieve a more natural appearance for the virtual objects, we made it possible to add new vertices on the edge of a surface, which means that the new vertices may be located at the intersection of multiple surfaces. However, this creates a new problem. Because triangulation is done for each surface, new vertices located on the edge are deemed to be located on one surface only. This will create holes between the surfaces. To avoid this problem, we need to add the vertices to the adjoining surfaces as well. To do this, follow the below steps:

(2-1) First, check the location of each vertex.

(2-2) Next, if a vertex is located at an intersection, add it to the adjoining surface(s).

The result of this step is shown in Fig. 5c.

(3) Divide the new surfaces using Triangle:

(3-1) Convert the virtual object's surfaces to 2D using step (0a).

(3-2) Divide the surfaces using Triangle: Unlike step (1-2), this time the surfaces are divided using existing vertices only. The newly divided surfaces' data and edges are then stored in the corresponding .ele and .edge files, respectively.

(4) Replace old surfaces and edges with new ones:

(4-1) List new surfaces by loading the data from the .ele files created in step (3-2).

(4-2) Delete the old surfaces and replace them with the new surfaces.

(4-3) List the new edges by loading the data from the .edge files created in step (3-2).

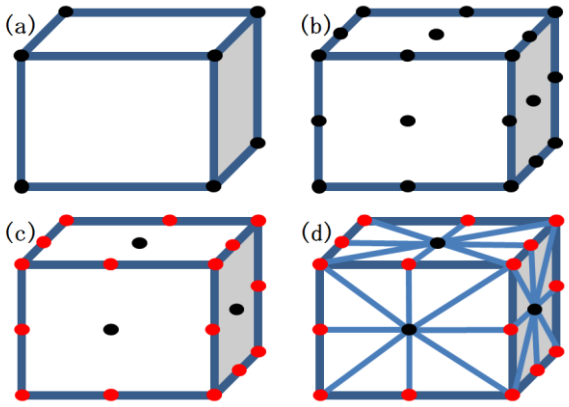

Fig. 5 Illustration of Triangulation: (a) Original data, (b) New vertices added, (c) Vertices added to adjoining surface(s), (d) Triangulated object

(4-4) Delete the old edges and replace them with the new edges.

The result of this step is shown in Fig. 5d.

(5) Calculate the normal vector for each surfaces and vertices.

An example of a divided virtual object is shown in Fig. 6.

4.2.2 Calculation for Deformation

In our system, users manipulate the shape of virtual objects by hitting them with HammerDevice. In addition, we use the edge of a real table to help users distinguish when they can dent or bend virtual objects. We display a semi-transparent yellow plane along the edge of the table as a marker, as shown in Fig. 7. When a virtual object intersects the plane, it is implied that the object is placed on the edge of a platform (in this case, a table), and users can bend the virtual object by hitting the protruding part. In contrast, to make a dent on a virtual object, users must make sure that the virtual object does not intersect the yellow plane before hitting it with HammerDevice.

In realizing shape manipulation functions, we need to consider all the parameters explained in section 2. Note that our goal is not to simulate real-life shape manipulation, but to propose an intuitive method for manipulating the shape of virtual objects. Thus, we do not implement a physics engine to simulate shape changes. However, we still need to make the changes in shape believable to some extent. The algorithms for both denting and bending are explained below in detail.

(1) Denting

When a virtual object is not placed on the edge of the table, users can make a dent in the object by swinging HammerDevice onto it. In particular, when HammerDevice is swung, the system determines whether the head of HammerDevice collides with a virtual object. If it collides, the system uses an accelerometer embedded in HammerDevice to determine whether it is swung with sufficient speed. If the speed is high enough, the system will determine the vertex that is closest to the head of HammerDevice. The vertex is then set as the central point for making the dent.

Because the system always keeps track of the devices' positions, we can determine the vector of the swinging direction by subtracting the current position from the position in the previous frame. This vector is stored as $\overrightarrow{dir}$. By dividing the length of this vector l with the time lapse between the previous and current frames t, we can determine the swinging speed v:

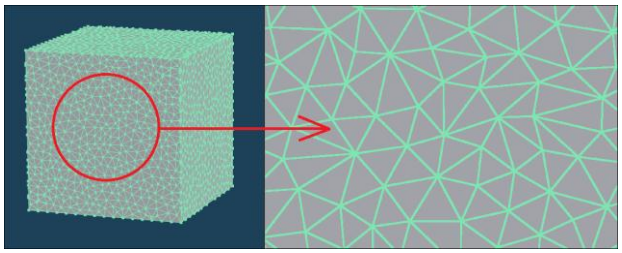

Fig. 6 Example of a triangulated object

Fig. 7 Yellow plane marking edge of table

$$v = \frac{l}{t} \quad (2)$$

The swinging speed is then used to calculate the size and depth of the dent. By multiplying the speed by the coefficient α, we determine the size of the dent:

$$r = \alpha \times v \quad (3)$$

Here, r indicates the radius of the dent. All vertices whose distance from the central point is less than r will be moved. How far and to which direction the vertices are moved is calculated using Eq. (4).

$$\overrightarrow{disp} = \left(1 - \left((\sin\left(\frac{d}{r} \times \frac{\pi}{2}\right)\right)\right) \times \overrightarrow{dir} \quad (4)$$

Here, $\overrightarrow{disp}$ is the vector of each vertex's displacement. Note that both r and $\overrightarrow{dir}$ are included in the equation. The idea is that a vertex's displacement is largest when the vertex is the central point. The further a vertex is from the central point, the smaller the displacement will be. As for $\overrightarrow{dir}$, the displacement of the vertices is adjusted to the direction in which HammerDevice is swung. Similar to operations in real life, the shape of a dent differs depending on whether it is hit perpendicularly or diagonally.

Finally, we calculate the new positions $\overrightarrow{P_t}$ for each vertex by adding the current position of the vertex $\overrightarrow{P_0}$ to the displacement vector $\overrightarrow{disp}$:

$$\overrightarrow{P_t} = \overrightarrow{P_0} + \overrightarrow{disp} \quad (5)$$

(2) Bending

When a virtual object is placed so that it intersects the yellow plane, users can bend it by hitting the protruding part with HammerDevice. To bend a virtual object, we first calculate the intersection line between the virtual object and the yellow plane. Next, using Eq. (6), for each vertex, we determine the closest point between the vertex and the intersection line and then calculate the distance between them:

$$\vec{D} = \vec{C} - \vec{V} \quad (6)$$

In this equation, $\vec{D}$ indicates the distance, $\vec{C}$ indicates the closest point, and $\vec{V}$ indicates the position of the vertex.

Using $\vec{D}$, we can determine whether a vertex is protruding from the table. To do this, we calculate the dot product of $\vec{D}$ and the normal vector of the yellow plane $\vec{N}$. If the dot product is less than 0, we consider the vertex to be protruding from the table:

$$\vec{D} \cdot \vec{N} < 0 \quad (7)$$

To create the impression of an object being bent, we rotate the vertices that are considered to protrude around the intersection line. How far a vertex is rotated is determined by its distance from the intersection line and the swinging speed of HammerDevice. We calculate the rotation angle using Eq. (8):

$$angle = \beta \times \vec{D} \times \frac{\pi}{2} \times v \quad (8)$$

Here, β is the coefficient and v is the swinging speed.

4.3 Interaction

4.3.1 TweezersDevice

TweezersDevice can be used to pick up and move virtual objects, just like using real-life tweezers. To help users more easily recognize the relative positions of TweezersDevice and virtual objects, the tip of TweezersDevice is indicated by a white sphere. To pick up a virtual object, users first have to make sure the white sphere collides with a virtual object and then pinch the object with TweezersDevice. By moving TweezersDevice while pinching, users can move virtual objects. The operations are shown in Fig. 8.

For operational feedback, when TweezersDevice touches a virtual object, a sound effect is played and a vibration motor is vibrated to alert the user. In addition, the color of the virtual object changes to green. These signals are provided to indicate that the virtual object can be picked up. To indicate that a virtual object is being moved, its color changes to blue.

Furthermore, when an object is selected, we utilize a braking system that uses a solenoid to provide force feedback to users. This gives users the feeling of

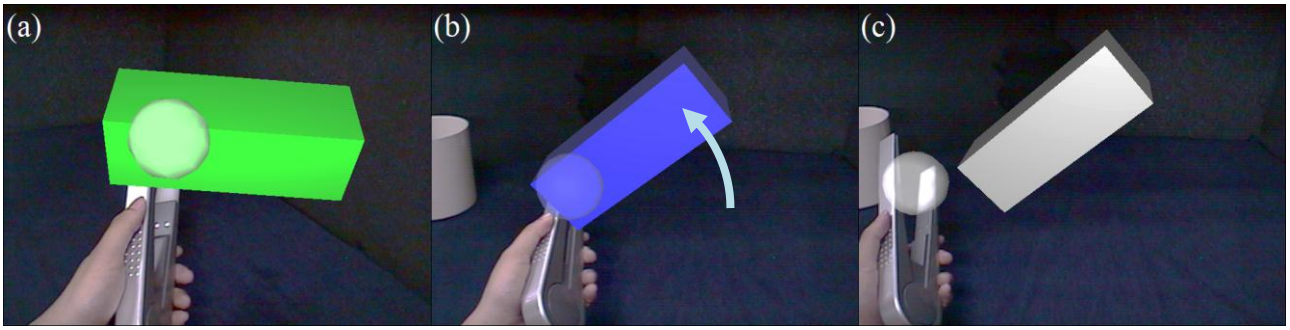

Fig. 8 TweezersDevice operation. (a) Pick (b) Move (c) Release

pinching a real object.

TweezersDevice can also be used to remove virtual objects. We map a virtual trash bin onto a real trash bin located on the table. By dropping virtual objects into the trash bin using TweezersDevice, users can remove unwanted objects. For operational feedback, when a virtual object collides with the trash bin, its color changes to red to indicate that the object can be removed. We also play a thrashing sound effect when the user releases the object to indicate that the object has been removed.

4.3.2 Knife/HammerDevice

In this study, Knife/HammerDevice is used mainly to manipulate the shape of virtual objects. Because shapes are manipulated using the hammer tip, we will refer to this device as HammerDevice.

(1) Denting

To make a dent on a virtual object, users first have to place the object anywhere except on the edge of the table. By hitting the virtual object with HammerDevice, users can make a dent in it. The operation is shown in Fig. 9a.

For operational feedback, when HammerDevice is swung and collided with a virtual object, a vibration motor is vibrated to indicate collision. In addition, we also play the sound effects of a metal object being hit to indicate that a virtual object has just been hit.

(2) Bending

To bend a virtual object, users first have to place the object on the edge of the table with the part that is to be bent extending beyond the edge of the table. By hitting the protruding part with HammerDevice iteratively, users can bend the object as much as they like. The operation is shown in Fig. 9b.

For operational feedback, as in denting, a vibration motor is vibrated when HammerDevice is swung and collided with a virtual object. In addition, the sound effect used in denting is also played to indicate that the object was hit.

5. Informal User Study

5.1 Objective

We conducted an informal user study to verify the intuitiveness of our proposed method. We also plan to improve our system based on user comments.

5.2 Procedure

Using the system that we developed, we let the participants manipulate the shape of virtual objects. Five people participated in the study, three male and two female students of Ritsumeikan University with an average age of 23 years. All of the participants had no experience whatsoever with any of our devices.

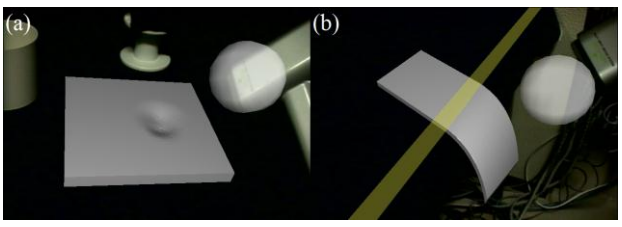

Fig. 9 HammerDevice operation: (a) Denting (b) Bending

First, we gave a short demonstration on how to manipulate the shape of virtual objects using our devices. Next, we let the participants use our system. We did not set any time limit, and the participants could freely ask questions while using our system. After the participants finished, we asked them what they thought about our system and the proposed method of operation.

5.3 Results and Discussion

Examples of the results produced by the participants are shown in Fig. 10.

Based on our observation, the participants were able to understand how to operate our devices after seeing the demonstration. When the participants held the devices, they started to bend and dent virtual objects without hesitation. However, on one or two occasions, the participants failed to manipulate the shape of virtual objects even after swinging HammerDevice. We assume that because the participants were not yet familiar with our devices, they did not swing HammerDevice with sufficient speed, which prevented the operations from being executed. We believe this problem does not need to be addressed specifically because the participants quickly learned that they need to swing the device with more speed.

We received several positive and negative comments from the participants. Many of the participants gave positive feedback such as “The deformation is nice” or “The virtual objects bent smoothly, just like real metal. The objects can even be bent after being dented, and the deformation is still as expected.” However, one participant found it peculiar to manipulate the shape of big and small virtual objects with the same force. We assume this is because in real life, small objects tend to be thought of as tough, and therefore hard to manipulate. This is something we did not expect, and we will try to look into this problem in more detail. As for other comments regarding the appearance, one participant commented that the texture resembles copper although the color is different. We admit we did not concentrate on the appearance of the objects. We will consider on improving it for the future work.

Regarding the operations, all the participants said that they are intuitive because they resemble operations in real life. However, some of the participants complained that it is hard to determine when to stop swinging HammerDevice. Although there are vibration and sound effects to indicate that a virtual object is hit, force feedback is lacking. Therefore, the participants have to guess when to stop, and this makes them uncomfortable. To solve this problem, we need to implement some type of force feedback mechanism. As for other comments regarding the operation, one participant said that he felt strange when making a dent because the virtual object is floating. To solve this problem, we thought of implementing gravity mode in

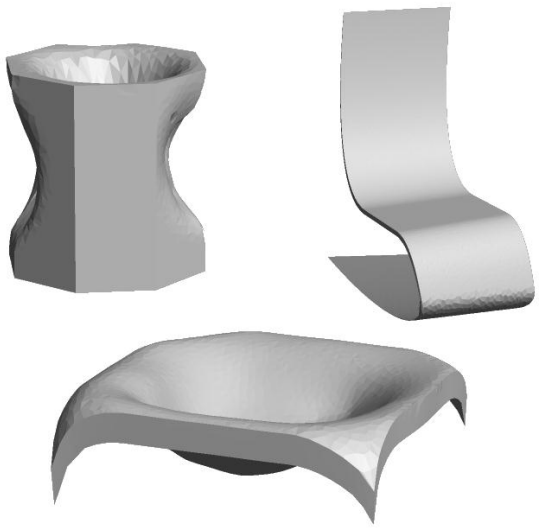

Fig. 10 User results

which virtual objects will stay on ground instead of floating. We think this is an interesting idea and will review it.

We received several negative comments on the devices, mainly about HammerDevice being heavy. One participant complained, "I do not understand the point of making the device so heavy. While other devices are relatively light, HammerDevice loses its merit when being used for mixed reality systems." In the previous work, we intentionally made the tip of HammerDevice heavier to give users a more lifelike feeling when holding the device. Unexpectedly, it was not popular among the participants.

Overall, the participants looked like they enjoyed manipulating virtual objects with our devices. In addition, from the comments received, we can say that our proposed method of operation is easy to understand because it resembles real-life operations. Furthermore, none of the participants incorrectly dented or bent virtual objects while trying to do otherwise. From these results, we can conclude that our proposed method of operation is intuitive.

6. Conclusion and Future Work

We proposed an interesting and intuitive way of manipulating the shape of virtual objects using ToolDevice, a set of interaction devices that imitates real-life tools in order to help users recognize its function. We introduced a metalworking metaphor to our modeling system. First, we modified the structure of virtual objects from our previous modeling system. Next, we implemented a shape manipulation function with two methods for changing the shape of virtual objects: denting and bending. Imitating real-life operations, we used the edge of a real table to differentiate these operations. To make a dent in an object, users can place the object anywhere but the edge of the table. Hitting the object with HammerDevice will then create a depression on the object. To bend an object, users first have to place the object on the edge of the table, with the part that is to be bent extending from the table. By hitting the protruding part with HammerDevice iteratively, users can bend the object as much as they like.

We conducted an informal user study with five participants to confirm the intuitiveness of our method. All of them seemed to understand the operations after seeing a simple demonstration, and none of them incorrectly dented or bent virtual objects while trying to do otherwise.

In future work, we plan to add haptic feedback to HammerDevice, join both modes into a single system, and conduct a formal user study to evaluate its usefulness and intuitiveness.

REFERENCES

[1] D. Anderson *et al.*: "Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling", Proc. SIGGRAPH 2000, pp. 393 - 402, 2000

[2] Y. Huang *et al.*: "Easigami: virtual creation by physical folding", Proc. TEI 2012, pp. 41 - 48, 2012.

[3] J. Sheng *et al.*: "An interface for virtual 3D sculpting via physical proxy", Proc. GRAPHITE 2006, pp. 213 - 220, 2006.

[4] K. T. McDonnell *et al.*: "Virtual clay: a real-time sculpting system with haptic toolkits", Proc. SI3D 2001, pp. 179 - 190, March 2001.

[5] G. Wesche *et al.*: "FreeDrawer – A Free-Form Sketching System on the Responsive Workbench", Proc. VRST 2001, pp. 167 - 174, 2001.

[6] S. Schkolne *et al.*: "Surface drawing: creating organic 3D shapes with the hand and tangible tools", Proc. CHI 2001, pp. 261 - 268, 2001.

[7] A. Uesaka *et al.*: "TweezersDevice: A Device Facilitating Pick and Move Manipulation in Spatial Works", Adjunct Proc. UIST 2008, pp. 55 - 56, 2008.

[8] M. Otsuki *et al.*: "MAI painting brush: an interactive device that realizes the feeling of real painting", Proc. UIST 2010, pp. 97 - 100, 2010.

[9] Y. Takami *et al.*: "Daichi's artworking: enjoyable painting and handcrafting with new ToolDevices", SIGGRAPH ASIA 2009, pp. 64 - 65, 2009.

[10] T. Grossman *et al.*: "An interface for creating and manipulating curves using a high degree-of-freedom curve input device", Proc. CHI 2003, pp. 185 - 192, 2003.

[11] I. Llamas *et al.*: "Twister: a space-warp operator for the two-handed editing of 3D shapes", ACM Trans. on Graphics 2003, pp. 663 - 668, 2003.

[12] H. Choi *et al.*: "Free hand stroke based virtual sketching, deformation and sculpting of NURBS surface", Proc. ICAT 2005, pp. 3 - 9, 2005.

[13] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, Applied Computational Geometry: Towards Geometric Engineering, volume 1148 of Lecture Notes in Computer Science, pp. 203 - 222, 1996.