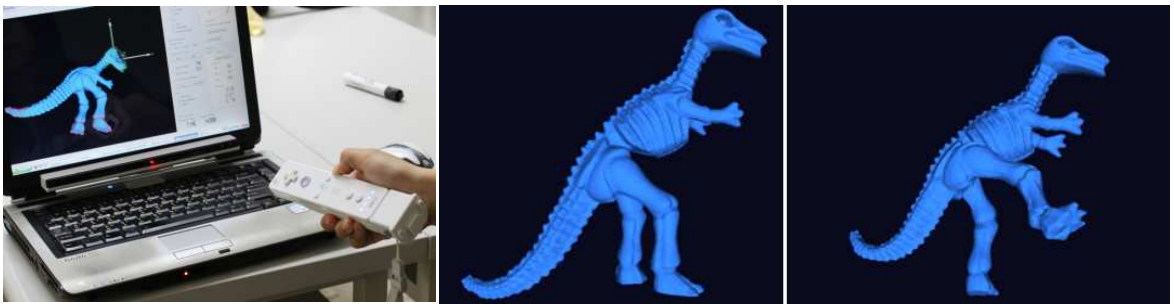
一套 Wii-MotionPlus 操作之三維模型編輯系統

陳愷軒, 鄭亦珊, 何星翰, 歐陽明
(國立台灣大學 資訊工程研究所 通訊與多媒體實驗室)

A Three Dimensional Mesh Deformation System Using Wii-MotionPlus

Chen Kai-Hsuan, Cheng Yi-Shan, Ho Hsing-Han, Ouhyoung Ming

(Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering, National Taiwan University)



(a) (b) (c)

Fig.1 (a) A photo of our 3D mesh deformation system using Wii-MotionPlus as the user interface. (b) and (c): the original and the resulting mesh deformed by our system.

Abstract: Mesh editing and deformation are critical in computer animation, such as in the motion of virtual characters in films and games. It needs lots of human resources to tweak the mesh to make the deformation look natural, and is often referred as “rigging” in animation industry. We propose a system that provides high level constraints to simplify complicated 3D deformation operations, and provide a novel user interface using Wii-remote MotionPlus to make the three dimensional control more intuitive. We find that our user interfaces can reduce the task completion time from 10% to 23%, according to three simplified experiments done by 10 subjects to deform the models in our user study.

Key words: mesh deformation; user interfaces; Wii-MotionPlus; skeletal animation

摘 要: 三維模型的編輯與形變在電腦動畫和模型建造上相當的重要，因其可運用在電影和電腦遊戲的角色編輯上。傳統上，調整模型形變的過程需要投入大量的人力與時間。因此，在這篇論文中我們提出一個系統由兩個方面來加速這個過程。第一，提出一個可以即時操作且盡量維持模型體積不變的形變演算法; 第二，提出新一套用 Wii 動感強化器作爲輔助的使用者介面。在我們的使用者測試中, 透過簡化的三項實驗, 發現此新型介面確實能幫助使用者縮短達 10%至 23%工作完成所需之時間。

关键词: 即時模型形變; 人機介面; Wii 動感強化器; 基於骨架之動畫
中图法分类号: TP399 **文献标识码**: A

1 Introduction

Making a 3D model to deform naturally is essential in 3D animation and modeling, which is important in game and movie industry. Traditionally, it's achieved by lots of animators' tweaks. Because this work is time-consuming and tedious, we observe that there are two directions that might help to speed up this process. First direction is a better user interface, and the second one is the help of high level deformation constraints.

The present modeling and animation software such as Maya and 3dx Max will control the rotation of a model through an arcball interface. Ideally, one should be able to rotate one, two, or three axes simultaneously. By dividing the screen into some parts, a user observes a 3D model in different direction in the same time and can use the mouse to modify the model in one of the divided screens. Because the mouse is basically a two dimensional device, the mapping between a mouse and the three dimensional rotation is a 2D to 3D mapping. Therefore, to avoid this constraint, we propose a new user interface that provides a 1-1 mapping of three-axis rotation by exploiting the Wii Remote MotionPlus. This interface directly maps the user control to 3D rotation, and that means, the model is rotated to the same direction as the Wiimote that a user holds. Moreover, translation can able be controlled simultaneously during the three-axis rotation by our interface, which is difficult for the traditional mouse and keyboard interface.

Wiimote, abbreviated from Wii Remote, was released by Nintendo in the end of 2006. This controller contains a three-axis accelerator that can sense the acceleration in three axes. Its drawback is that it can't sense the uniform motion without acceleration, hence we can't properly detect the real 3D direction of Wiimote. Recently, Nintendo released the extension of Wiimote, Wii MotionPlus. Wii MotionPlus contains a gyroscope. By connecting this extension to the original Wiimote, we can read the relative rotation of Wiimote, even under the motion of uniform speed. After we calibrate it, we can calculate the exact direction of Wiimote relative to original direction, and make precise control of three dimensional rotations possible. Therefore, we utilize Wiimote and its extension, Wii MotionPlus, to build our user interface.

Modifying an existing model is much easier than creating a new model, and that's the reason why deformation method is worth to research. Many methods have been proposed to deform a 3D model.

Besides the user interface, we build a 3D mesh deformation system that provides convenient editing operations. In the editing process, users usually anticipate a model to deform according to certain physical rules, like preserving rigidity and geometric details. In our system, users just need to fix certain vertices in the mesh and dragging other vertices in the mesh, and the intermediate parts of the mesh will be deformed automatically based on constraints. Our implementation is based on Sorkine et al. [2, 3].

There is a standard editing procedure for our system. First, a user have to assign two parts of a model, the fixed parts and a moving part, by selecting certain desired vertices. Our system provides a handle of the fixed part, and a user can rotate and/or move the handle to drive the moving part. The position of those vertices between the fixed and moving parts will be optimized during the editing process by our system.

As conclusions, we present an interactive deformation system uses high level constraints, including details and rigidity constraints to speed up the tedious adjustment of animators, and an intuitive user interface by Wiimote MotionPlus. The whole deformation system not only makes the deformation process more intuitive but also reduces the time of modifying a model.

2 Related Work

In these section, we will discuss the works relate to mesh deformation and user interface respectively. And the related literatures of the user interface include two classes of works. First, the works related to Wii Remote. Second,

the works related to the user interface of mesh editing.

2.1 Mesh deformation

The literatures of mesh deformation and editing can roughly be separated into two approaches, surface-based approach and space-based approach. Surface-based methods formulate the deformation as a mesh membrane functional, and space-based approach uses the other coordinates to control the deformation.

For our system, interactive manipulation is needed, therefore linear method is suitable. Many linear surface-based methods have been proposed in recent years. They are attractive because of that they can solve the problem efficiently, and this characteristic is very suitable for interactive application. In addition, we can get a global minimum in a linear system under an appropriate boundary condition. Moreover, most linear methods formulate the deformation energy as a smooth function, hence a small perturbation will not cause large changes in the model surface.

However, linear methods are not perfect. Since rotation is nonlinear, the deformation is inherently a non-linear problem. Linear methods can only provide an approximate result, and the results under large rotation are not usually satisfactory. This problem can be solved by dividing the large deformation into small ones, although it makes the work heavier.

The following are various linear deformation methods. Laplacian coordinates is used in [1, 2]. Naïve Laplacian [1] only keeps translation-invariant, and Sorkine et al. [3] improve it to small-angle-rotation-invariant. Lipman et al. [4] introduced a linear rotation-invariant coordinates that retains model geometry even under large rotation angle. Botsch et al. [5] presents a system that integrates multiresolution mesh editing framework, and provides a 9-dof handle for users. Yu et al. [6] introduces a deformation approach based on Poisson equation. It utilizes a specially designed handle to manipulate the gradient field of origin mesh, then the boundary condition of the constrained vertex are propagate to the vertex in the influence region. In the end, the deformed mesh is obtained by solving the Poisson equation.

Reconstruction of mesh geometry in above methods all involves solving a sparse linear system. Using specially designed sparse linear solver as in [7], we can factorize the matrix before deformation. After pre-factorization, the linear system can be calculated by back-substitution, which is more efficient.

In recent years, many robust nonlinear methods have been proposed. Primo system [8] builds a prism to each face in a mesh by extruding a changeable offset along the vertex normal, and the deformation energy simulates that there are several springs between neighboring prisms. The height of prisms can be adjusted by users to control the surface's stiffness, which means the resistance of bending. Mesh puppetry [9] provides five high level constraints to preserve mesh surface details, balance, length, rigidity, and joint limit.

Our system is based two deformation methods. We use the linear method [2] to get the rough initial guess, and then refine the initial guess by the nonlinear method, as-rigid-as-possible surface modeling [3]. These computation can be executed in interactive rate.

2.2 User interfaces

The Nintendo Wii-controller, Wiimote, has become one of the most common input devices after Wii was released by Nintendo, in 2006 winter. Combining its reasonable price and potential ability of three-dimensional controlling, Wiimote became an excellent option to be the interface of our 3D mesh editing system. By gesture recognition [10], Wiimote provides more emerging experience than traditional keyboard and mouse.

Wiimote has been developed as input interface for many applications. Sreedharan et al. [11, 12] used Wiimote for navigation of virtual 3D environments. Shiratori et al. [13] attached multiple Wiimotes to a user's legs and head

to control a virtual biped character. A user's movements, include walking, jumping, and running, are mapped to a virtual character. This mapping enhances the emergence of users.

Since traditional interfaces, such as mouse and tablet, are 2D devices, it's a challenge to design a 3D geometric modeling and deformation interface. The most common modeling software, like 3ds Max and Maya, still use mouse as the controlling device. Some works attempt to achieve 3D modeling conveniently using 2D interface. Teddy system [14] is a tablet-based sketching interface for 3D freeform design, and it generates a 3D model that fits common people's prediction. Teddy system is enough to design rough models in concept design or playing, but it's not adequate to design complex models. FiberMesh system [15] is a system similar to Teddy system, but adding more modeling operations. However, a user is not able to design sophisticate models using FiberMesh system.

2D-sketch-based interfaces [16, 17] are another approach for mesh editing systems. Nealen et al. [12] modifies a model by a handle, which is determined by silhouette selection or cropping, or directly drawing on the model surface. In [17], users draw a line or curve on a model, where the curve acts like a bone of this mesh, and bending this curve will drive the neighboring mesh to deform. Gingold et al. [19] presented a shading-based surface editing system that allows a user to modify a shape by changing its rendered image. The 2D user input is translated to 3D model transformation. In some modeling cases, this system is easier to achieve the deformation goal than standard deformation approaches.

3 Mesh deformation algorithm

Our goal of mesh deformation algorithm is to preserve geometric details and rigidity during modeling operations, and this algorithm should be fast enough for user to manipulate interactively. This goal is achieved by an iterative two-step method based on [2, 3]. Generally speaking, during each deformation, we first compute a rough deformation as an initial guess using Naïve Laplacian coordinates [2], and iteratively refine it using [3].

3.1 Initial deformation

It's fundamental to preserve geometric details during modeling operations, and we achieve this goal by converting vertices from absolute coordinates to differential coordinates. Differential coordinates captures the relative relation, known as details of mesh, between vertices. Absolute Euclidean coordinates changes during transformations, such as translation, rotation and scaling, nevertheless, differential coordinates like Laplacian coordinate retains its value during transformations. For free-form deformation, we set an object function that penalizes changes of value of Laplacian coordinates in all vertices in the targeting mesh, and finding the best mesh after deformation by minimizing this object function.

The Laplacian representation of a vertex I, w_i, is shown as below, and N(i) represent one-ring neighboring vertices of vertex i in mesh M.

$$w_i = V_i - \sum_{j \in N(i)} \frac{1}{|N(i)|} V_j$$

The transformation from absolute Euclidean coordinates to Laplacian coordinates, a forward transformation, can be represented in matrix form, $L = I - DA$. I is an identity matrix, D is a diagonal matrix with $d_{ii} = 1/|N(i)|$, and A is the adjacency matrix of mesh M. If we write all vertices in mesh M in a vector V, we can get $LV = W$. Assume we write all vertices in absolute coordinates in a vector V', we can solve $LV' = W$ to get the best position. We solve this equation in three dimensions separately.

In addition, matrix L can be very large when there are many vertices in the mesh. Since the number of one-ring neighbor of a vertex is usually small, matrix L is very sparse. Therefore we solve this constraint by a linear system solver library [7] which is specifically designed for solving sparse linear systems.

3.2 Refinement

For an input model, we treat each vertex and its one-ring neighboring vertices as a cell. The model is kept as rigid as possible by seeking the best, which means as rigid as possible, transform for each cell during deformation, and the overlapping of these small cells prevent surface from stretching.

For cell i which center is vertex i, position p_i and one-ring neighbor N(i), we want to find the transformation R_i as the following:

$$p_i' - p_j' = R_i(p_i - p_j), \text{for each } j \in N(i)$$

For a cell i, we can formulate the following energy function:

$$E(C_i, C_i') = \sum_{j \in N(i)} w_{ij} \left| (p_i' - p_j') - R_i(p_i - p_j) \right|^2$$

For the whole mesh, we can obtain the following energy function:

$$E(S') = \sum_{i=1}^{n} E(C_i, C_i') = \sum_{i=1}^{n} w_i \sum_{j \in N(i)} w_{ij} \left| (p_i' - p_j') - R_i(p_i - p_j) \right|^2$$

Intuitively, different-size triangles in a mesh should be given different weights, and we adopt cotangent as the weights.

An iterative method is adopted to minimize this non-linear system. There are two steps in each iteration, and these two steps are both linear and easy to solve. In the first step, we treat new mesh vertex positions p' as constant and solve R for each cell. The second step, we treat R as constant and optimize all vertex positions of the new mesh. The two-step optimization keeps going until the error is smaller than the threshold.

In the first iteration, we use the mesh which is obtained in previous section. Then the mesh is refined in every iteration. In practice, the optimization accomplishes in less than three iterations, and our system runs at interactive rate when the region of interest (ROI) contains 3k vertices.

4 User Interfaces

MotionPlus

Wii-MotionPlus contains a tiny gyroscope inside, allowing more accurate capture of complex motion. MotionPlus Device can get angular momentum from three axes which the original controller is lack of.

The original Wiimote contains an optical sensor and an accelerator device to control user's motion. Optical sensor (Infrared Rays Sensor) can catch the infrared rays emitted from the infrared rays bar, which is usually put on the TV when playing Wii. Wiimote reads the rays' position to get

Fig.2 Wii MotionPlus, an expansion device for Wiimote.

the pointer position on the screen, which is often used in game interface and shooting game.

Wii accelerator device provides a way to capture acceleration from three axes. In most sport games the accelerator captures the strength of user's action, like striking the balls, and simulating the action in the game. In most games it seems to work well, but actually the precise orientation is still missing; it works whether we strike the ball by waving from the right to the left or from the top to the bottom. The accelerator is good when asking for the strength of the motion only, but bad in detecting the accurate motion of the Wiimote controller.

In 2009 Nintendo announced Wii-MotionPlus. The MotionPlus device contains a small gyroscope, which is often placed on ships to maintain the orientation. A mechanical gyroscope is essentially a spinning wheel or disk whose axis is free to take any orientation. MotionPlus takes advantage of the function and helps us to detect the precise motion of Wiimote in any orientation.

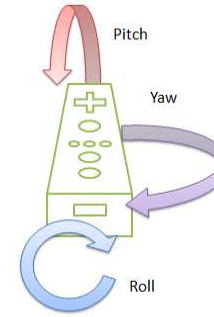


Fig.3 Pitch, Roll, Yaw rotation on Wiimote. These rotations are local and used to describe the rotation in each single time interval.

For the Wiimote controller local rotation, Pitch is the rotation by X axis, Roll is by Y axis, and Yaw is the rotation by Z axis. The following X, Y, Z axes are relative and local coordinate of Wiimote itself, so it is more precise to describe them as Pitch, Roll and Yaw.

The main feature of Wii-MotionPlus is that it returns the local changing amount of Pitch, Roll, and Yaw at every single time interval, but not the global angle of rotation. Namely we have to accumulatively sum the values to guess the global 3D rotation. This property may trigger some accumulation error, and certain calibration method must be involved.

Initialization:

At the beginning we must correct errors from tiny vibration. Even if we put the controller on the table, the small signal errors may still sum up to a visible amount. When initializing, we choose the average value in 300 clock time to be the base value, and every forward movement value will be normalized by the base value. In some new Wii-Motion-Plus- based games announced by Nintendo, the players also need to calibrate the controller by holding it statically for a moment to ensure the quality of the game.

Getting Wiimote Orientation by MotionPlus:

After initialization we can start summing the angular momentum of Pitch, Roll and Yaw. It is a problem to directly map the Pitch, Roll and Yaw to three axes rotation. The x-y-z coordinates system acts globally but the roll-pitch-yaw coordinates are local; in addition, some coordinate transformation problem and 3D rotation problem exists. We have to map the local rotation to global x-y-z rotation. To avoid coordinate transforming problem, we have to rotate the coordinate base on Pitch-Roll -Yaw angle for every single movement.

Although at the beginning we calibrate the controller and get average error when it is placed at static place, there are still small errors exist due to the unstable signal. It will cause an obvious drift in the long run. We need a threshold to filter the small movement to avoid the drift, but in this way some tiny movement may go undetected - such as the moment wii-controller starts to accelerate or slow down, and therefore accuracy goes down after a period of time. There is no effective way to ignore tiny signal errors when silence and keep those tiny movements when moving as while, and the situation will get even worse in accumulative system. One Solution is to do calibration frequently, like in certain Wii games the players have to re-calibrate the controller orientation in every round. Our solution is to re-calibrate the Wiimote controller by its own accelerators.

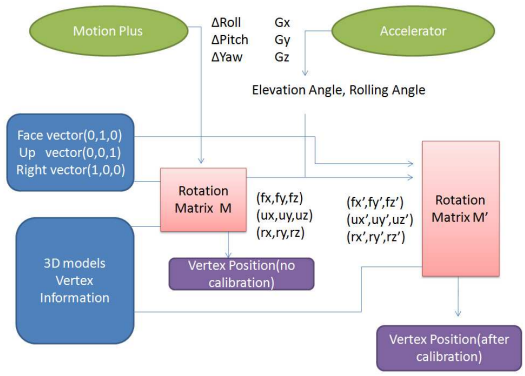


Fig.4 The flow chart of self-calibration. The new 3D models vertex information can be achieved by MotionPlus directly. To get more precise result, we can use the information from the Accelerator to do self-calibration.

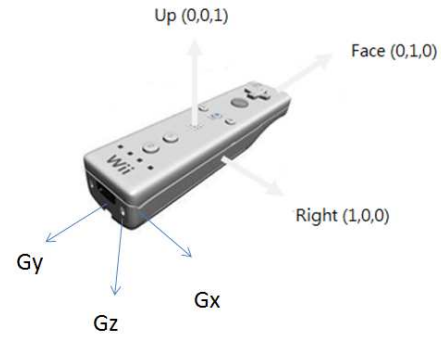


Fig.5 We assume that the controller is placed flat and pointing the middle of the screen at the beginning. The starting face vector is (0,1,0), the up vector is (0,0,1) and the right vector is (1,0,0). When the controller is at rest, the values from the Accelerator can been regarded as the sub-vector of the gravity in three axes– Gx, Gy and Gz.

Self-Calibration

When static, the controller always gets a pure gravity **G**, and can be obtained from the accelerator from three axes **Gx**, **Gy**, and **Gz**. Different orientation results in different Gx, Gy and Gz values. Considering the vector (Gx,Gy,Gz) is a gravity vector ,we can find the angle between the vertical vector(0,0,-1) and gravity direction, thus we can adjust the orientation of controller. We can't judge the global orientation by the accelerator only, because there are infinite many orientation solutions which have the same gravity measurement when rotating around z-axis (vertical axis). Namely, once we get a set of gravity measurement, every possible solution of each point forms a circle on x-y plane (horizontal plane). As a result, we keep the face vector of controller on horizontal plane and adjust the elevation angle and rolling angle. Knowing the elevation angle and rolling angle, we can re-estimate axes vector and find the new transformation relation.

We start our self-calibration with the initial result of Motion-Plus. Let the rotation matrix derived from MotionPlus be ***M***. Every point multiplies the rotation matrix and gets a base value.

We use the Gy value to compute elevation angle. Gy is the force passing through the head and the bottom of the controller, that is, the vector that parallels to the controller. Once we change the elevation angle, the Gy value changes in the meantime no matter what the controller's orientation is. We set the elevation angle to adjust the controller's face direction vector(fx,fy,fz) , save the fx,fy value computed by MotionPlus, and adjust the fz value by accelerator:

$$\text{Face Vector}(fx, fy, fx) = [0,1,0]\ \text{M}$$
$$fx' = fx, \quad fy' = fy$$
$$\text{Elevataion angle} = \cos^{-1}(-Gy)$$
$$fz' = |fx', fy'| \ * \ \tan(\text{Elevation\_angle})$$

The second step is to re-estimate the rolling angle, which is a relative value in controller's local coordinate. We use the controller's upward direction vector to describe it. To find the accurate "up" vector **(ux, uy, uz)**, we use

the result of the "face" vector, rotate 90 degree along the "right" vector (like elevate the vector), and finally rotate along the rolling angle around face direction. The rolling angle can be obtained from the ratio of **Gx** and **Gz**. Knowing that the $Gx^2 + Gz^2$ value will be a constant when rolling, we can compute local rolling angle:

$$\text{Up Vector}(ux, uy, uz) = (fx', fy', fz') \text{ rotate } 90° \text{ around } (-fy, fx, 0)$$

$$\text{Rolling angle } = \sin^{-1}\left(\frac{Gx}{|(Gx,Gz)|}\right)$$

$$(ux', uy', uz') = (ux, uy, uz) \text{ Rotate Rolling angle around } (fx', fy', fz')$$

At last, we use the "face" vector and "up" vector to get the "Right" vector.

$$(rx', ry', rz') = (ux', uy', uz') \text{ rotate } 90° \text{ around } (fx', fy', fz')$$

Now we know the face, up and right vectors of the controller by MotionPlus and Accelerator. At the initial calibration, the controller is placed flat toward the screen; that is, the face vector, the up vector and the right vector are (0,1,0), (0,0,1) and (1,0,0). We can then form a rotation matrix between the old and the new vectors:

$$[F|U|R]\, M' = [F'|U'|R']$$
$$M' = [F|U|R]^{-1}[F'|U'|R']$$

Hence we get re-estimate new rotation matrix **M'**.

Self Calibration method can solve the drifting problem, but two issues must be considered: first, calibration can correct the drift error only in two axes, while the drifting on x-y plane cannot be solved. Second, the accelerator detects pure gravity when static but heavy pulses when waving the controller. The self calibration algorithm only works when computing mere gravity, and therefore it fails when the user is waving the controller. Consequently, in our work we only do self calibration when user stops waving the controller.

5 Experiment and Result

With the above deformation system which uses Wii Remote and MotionPlus as the user interface, we conducted three experiments to demonstrate the advantages of our interface over the mouse.

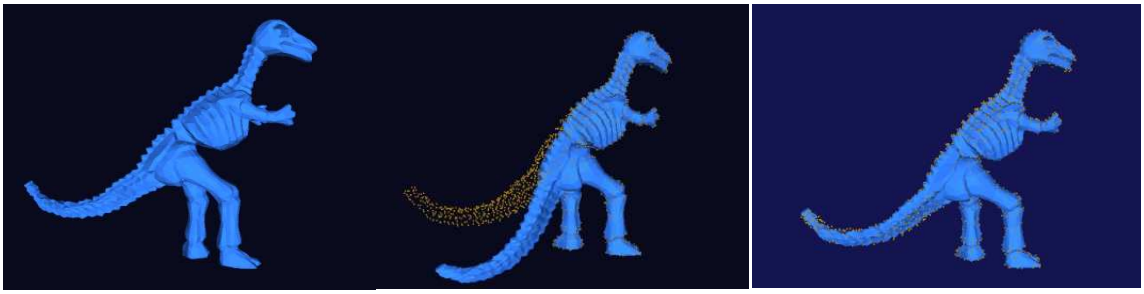
We found ten users to participate our experiments as subjects, and designing three deformation tasks for subjects to complete. All of the tasks involved both translation and rotation operations. Before the experiment, we would take fifteen minutes to train the subject to be familiar with these two interfaces, and let them to exercise for a simple task. Then the subject was requested to complete these three tasks with traditional mouse interface and Wii MotionPlus interface, respectively. The completion time in seconds for each task was recorded, as shown in Table 1.

	Task 1: Dinosaur	Task 2: Horse	Task 3: Armadillo
Completion time of mouse interface (seconds)	28.56	36.75	27.97
Completion time of our interface (seconds)	21.94	33.28	21.88
Speedup (%)	23.18	9.52	21.77

Table1. The task completion time of two different ways for three tasks.

Before each task, we would first demonstrate the target model to the subject (Fig.6(a)). During the experiment,

we displayed a 3D yellow point cloud (Fig.6(b)) to guide the user to achieve the desired goal. If the deformation is close enough to the goal, the background color would change to notify the user.

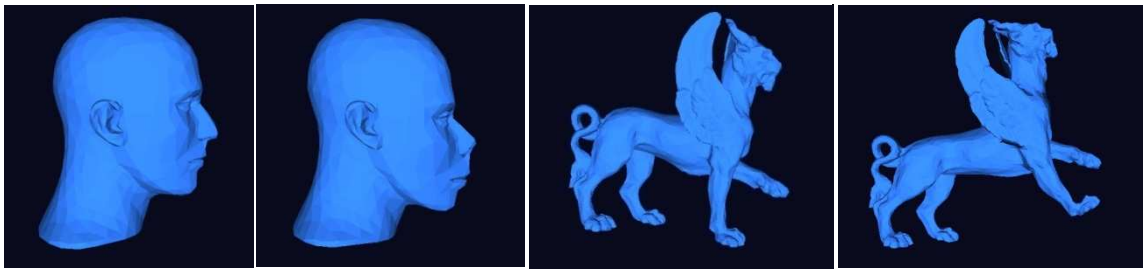


(a) (b) (c)

Fig.6 (a) The target deformation. (b) We guide the subject using yellow point cloud which shows the position of the target deformation. (c) The background color will be changed if the model is close enough to our target deformation.

We designed a questionnaire for users, which includes two classes of question - translation operation and rotation operation. The result indicates that the Wii-MotionPlus and mouse interfaces are competitive in translation operations, but Wii-MotionPlus interface is more intuitive in rotation operations.

DEMO: At the end of this section, we will put a few results in Fig.7, and our two-minute demo video can be found on YouTube by searching the title of our paper.



(a) (b) (c) (d)

Fig.7 (a) and (b): the original and the resulting mannequin mesh. (c) and (d): the original and the resulting feline mesh.

6 Conclusion and Future work

In this paper, we propose a new deformation framework. A cheap and affordable hardware, Wii Remote MotionPlus, is utilized to facilitate the operations of users. Besides the user interface, our deformation is based on a special-designed function, which attempt to retain rigidity of the given mesh.

In the user interface, we adopt Wii Remote and MotionPlus. This interface improves over the traditional 2D mouse interface, which lets the users be more immersed to the system. Besides, we designed an automatic re-calibration method with the help of an accelerator to overcome the accumulation error of Wii MotionPlus.

Our deformation algorithm is a cell-based iterative algorithm. At first, we will compute an initial guess using the naive Laplacian algorithm. Based on the initial guess, we compute the local rotations relative to the original mesh of each cell, and optimizing the mesh using these local rotations. The two-step computation is kept going until convergence, which is usually less than three iterations. Although the cells in this algorithm are surface-based, the experiment result is visually convincing.

In conclusion, our deformation system provides an innovative user interface. We integrate the Nintendo Wii

Remote MotionPlus to our system for 3D rotation manipulation that is 1-1 mapping. In the inner deformation loop, we adopt an algorithm that preserves rigidity during mesh deformations. And the high level rigidity constraint helps users to achieve the desired results easier and more efficiently.

In the future, we wish to adopt an algorithm that provides different levels of rigidity. In the user interface, we believe that the multi-touch technique is another possible option which is appropriate for common users, especially for children to have fun playing with.

References:

[1] Sorkine O, Cohen-Or D, Lipman Y, Alexa M. Laplacian Surface Editing. ACM International Conference Proceeding Series Vol. 71, 2004.

[2] Alexa M. Differential coordinates for local mesh morphing and deformation. The Visual Computer, 2003 – Spring.

[3] Sorkine O, Alexa M. As-Rigid-As-Possible Surface Modeling. ACM International Conference Proceeding Series, Vol. 257, 2007.

[4] Lipman Y, Sorkine O, Levin D and Cohen-Or D. Linear rotation-invariant coordinates for meshes. ACM SIGGRAPH 2005.

[5] Botsch M, Kobbelt L. An intuitive framework for real-time freeform modeling. ACM SIGGRAPH, 2004.

[6] Yu Y, Zhou K, Xu D, Shi X, Bao H, Guo B, Shum HY. Mesh editing with poisson-based gradient field manipulation.ACM SIGGRAPH 2004.

[7] Toledo S, Chen D, Rotkin V. TAUCS: a Library of Sparse Linear Solvers. Tel Aviv University, 2003.

[8] Botsch M, Pauly M, Gross M, Kobbelt L. PriMo: coupled prisms for intuitive surface modeling. ACM International Conference Proceeding Series; Vol. 256, 2006.

[9] Shi X, Zhou K, Tong Y, Desbrun M, Bao H, Guo B. Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. ACM SIGGRAPH, 2007.

[10] Schlömer T, Poppinga B, Henze N, Boll S. Gesture recognition with a Wii controller. Tangible and embedded interaction, 2008.

[11] 3D input for 3D worlds, Sreedharan S, Zurita ES, Plimmer B. OZCHI, Vol. 251, Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces, 2007.

[12] Schou T, Gardner HJ. A Wii Remote, a game engine, five sensor bars and a virtual reality theatre. OZCHI, Vol. 251, Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces, 2007.

[13] Takaaki S, Jessica KH. Accelerometer-based user interfaces for the control of a physically simulated character. ACM Transactions on Graphics, 2008

[14] Igarashi T, Matsuoka S, Tanaka H. Teddy: a sketching interface for 3D freeform design. ACM SIGGRAPH, 2007.

[15] Nealen A, Igarashi T, Sorkine O, Alexa M. FiberMesh: designing freeform surfaces with 3D curves. ACM Transactions on Graphics, 2007.

[16] Botsch M, Pauly M, Gross M, Kobbelt L. A sketch-based interface for detail-preserving mesh editing. ACM International Conference Proceeding Series; Vol. 256, 2006.

[17] Kho Y, Garland M. Sketching mesh deformations. ACM SIGGRAPH, 2007.

[18] Gingold Y, Zorin, D. Shading-based surface editing, ACM Transactions on Graphics, 2008.