Proceedings of 2012 IEEE
International Conference on Mechatronics and Automation
August 5 - 8, Chengdu, China

# A Data-driven 3D Vision Feedback System for Bilateral Rehabilitation Application

Xiang Kui[*1, *2], Guo Shuxiang[*1, *3], Song Zhibin[*1]

[*1] *Dept. of Intelligent Mechanical Systems Eng'g, Kagawa University, Hayashi-cho, Takamatsu 761-0396, Japan*

[*2] *School of Automation Wuhan University of Technology Wuhan 430070, China*

[*3] *College of Automation Harbin Engineering University Harbin, Heilongjiang, China*

xiang@eng.kagawa-u.ac.jp, guo@eng.kagawa-u.ac.jp, song@eng.kagawa-u.ac.jp

*Abstract* – **The patients in bilateral rehabilitation move their paretic limbs to follow the trajectory defined by imparetic limbs. The robot provides power only when the patients need assistance. Capturing the motion posture and feeding back to patients through audio or visual modes can raise training enthusiasm and activeness and obtain better rehabilitation effect. A mini even free vision feedback system is on demand when rehabilitation trainings need to be implemented at home. In this study a vision feedback system is developed based on open source software. The original 3D character is generated by a piece of special software Makehuman and exported as several optional formats such as obj, mtl, bvh and so on. These exported files are re-loaded in a graph interface built with wxWidgets and rendered with OpenGL. The motion sensors are fixed on the patient's limbs and the motion data are sampled to update 3D model rendering. The patients' motion posture is shown on a screen in real time to prompt their rehabilitation training. The new vision feedback system presented in this paper does not depend on any commercial software or expensive hardware. As a fully open source system, it is very appropriate to realize more economical rehabilitation trainings.**

***Index Terms –Vision Feedback, Open Source, 3D Model, Rehabilitation.***

## I. INTRODUCTION

Stroke is the leading cause of disabilities especially in developed countries because of the unhealthy living style. A great number of patients survive a first or recurrent stroke and suffer from hemiparesis which is the common motor deficit. Disabilities in the extremities severely influence daily activities. The previous studies have proved that the neurons are of plasticity and the motion functions can be recovered by iterative motor exercises. Thus more effective rehabilitation techniques except for therapists are looked forward to, and the prevail supplement is rehabilitation robots [1].

Most of studies in the past focused on the unilateral training in which the rehabilitation actions only performed on paretic limbs. Recently, a new suggested therapeutic technique is bilateral training which requires two limbs to synchronize in rehabilitation actions [2]. Bilateral training has two main advantages: the imparetic limb provides the training trajectory guiding the paretic limb; two limbs simultaneous training encourages the inter-limb coordination. Generating the trajectory by imparetic limbs is so convenient that independent training at home without therapists attending is feasible. The bilateral movement can activate the undamaged hemisphere to promote neural plasticity and enhance the therapy for damaged hemisphere. A randomized, controlled trial involving 127 patients with upper-limb impairment receives intensive robot-assisted therapy, intensive comparison therapy, and usual care, respectively [3]. At 12 weeks, the robot-assisted therapy had not more advantages in motor function improvement; but over 36 weeks robot-assisted therapy improved outcomes as compared with usual care.

There are many robot researches on bilateral rehabilitation. Lum and *et al* [4] presented the mirror-image movement enable (MIME) robotic device and investigated the hypothesis that the robotic training improves muscle activation. Thirteen stoke survivors trained in MIME increased their reach extent and speed. Song and *et al* [5] designed an exoskeleton device with three active DoFs and four passive DoFs to assist the paretic limb on the elbow and wrist for stroke subjects. At the same time, a haptic device (Phantom Premium) was adapted on the intact side to guide the motion of paretic limb. The exoskeleton is portable and suitable for home-rehabilitation or remote rehabilitation. Mihelj and *et al* [6] proposed and evaluated a new paradigm for patient-cooperative control strategy. The patients try their best for reaching movements with their own trajectory, and the robot helps them only when the training task can not be completed. Such minimal intervention principle allows the patients' activeness unless their actions interfere with task specifications. Cai and *et al* [7] proposed a robotic training paradigms, assist-as-needed (AAN), to encourage the rehabilitation movement variability. Twenty-seven mice were collected to test the efficacy of robotic control strategies, and found that AAN window paradigm showed the highest level of recovery.

Virtual reality (VR) as a unique medium provides a functional, purposeful and motivating context for effective rehabilitation intervention. VR can produce simulated, interactive and multi-media environments and allow users to interact in real time with images modulated by motion sensor information [8]. The past studies have demonstrated the VR application in the treatment of stroke survivors [9]. Furthermore, augmented reality (AR) does not fully pursue the immerse sense like VR, and it provides an environment with the real and virtual objects coexisting in the real world [10]. AR is very useful to rehabilitation systems targeting daily living activities.

978-1-4673-1277-6/12/$31.00 ©2012 IEEE

Trlep and *et al* [11] designed a virtual flight simulator game using Unity3D software to enhance subject's motivation. In the game one of the planes represents the tracker's pose corresponding to the pose of the bilateral handlebars. Liu and *et al* [12] combined a dual video to meet rehabilitation training needs and made the motion analysis feed back to the patient in real time. Secolil and *et al* [13] researched on the visual target tracking while receiving adaptive assistance from an arm exoskeleton robot. They found that incorporating real-time auditory feedback of performance errors might improve clinical outcomes. Tao and *et al* [14] introduced a motion tracking system with vision and inertial sensors for home based rehabilitation. The experiment shows the system can track the 3D arm motion in real time with acceptable accuracy compared with the commercial marker based system. Brewer and et al [15] constructed a controllable visual distortion environment to amplify the imperceptible error of the force and position during training. They found that the visual distortion can be used to alter a patient's perception of therapeutic exercise under the assist of a robot.

Wu and *et al* [16] designed VR rehabilitation to enhance the effect of mirror therapy. The VR system evokes the patient's motor image and acquires motor abilities. Loconsole and *et al* [17] built a computer vision system recording the rehabilitation training with webcam and redisplaying in a screen with OpenCV. The reaching target is recognized in OpenCV and used as the control goal of the robot. Vergaro and *et al* [18] replicated the picture on the screen that included the predefined path, moving target and hand position to automatically adapt the man machine interaction forces. Duschau-Wicke and *et al* [19] presented a patient-cooperative strategy by showing a compliant virtual wall on a screen. Graphical feedback provides visual training instructions to encourage larger temporal variability and frees patients from the robot control. Wang and *et al* [20] integrated the visual error augmentation with the AAN training in robot-assisted rehabilitation system. The visual error augmentation heightens the patient's motivation to improve tracking accuracy.

Based on the above reviews, we can conclude that the vision feedback system play a crucial pole in AAN training paradigms besides the patient and robot. It is a close-loop system including a VR environment and patient. The traditional VR research focuses on game design. But the vision feedback system specializes in constructing a vivid 3D human model and driving it with motion data. A new vision feedback system oriented to rehabilitation application is design based on a professional human character and MTx sensors. We will present the details about how to generate, load and drive a 3D model in the following sections.

## II. Open Source GUI Programming

Graphical User Interface (GUI) is the core for a vision feedback system design. Selecting an appropriate GUI tool is not easy because there are so many choices in a Windows-prevailing age. Three important standards are list for choosing a GUI tool: open source, free, powerful enough but not cumbersome. Many tasks such as sampling, analysis and control are usually embedded in vision feedback systems and executed in background, which is different from one-fold game design. These background tasks have real time demand. The open source GUI can ensure foreground updating fully cooperate with background computing. The free GUI can cut down the cost of rehabilitation equipments and a mini even free vision feedback system is especially appropriate to home or remote rehabilitation. A powerful GUI tool can reduce the programming burden. But too powerful tool may be redundant because the vision feedback system is just a piece of lightweight software.

Taking Microsoft Foundation Classes (MFC) as an example, it is a very powerful GUI tool and can perfectly join in Windows operation system, but it is fully commercial and not open source. QT is another powerful GUI tool with commercial and free versions. It is open source and cross platform application better than MFC, but too cumbersome to deal with vision feedback programming. Fast Light Toolkit (FLTK) seems to a good choice, but it is so lightweight that not able to support networking, printing etc.

wxWidgets is considered as the most appropriate GUI for vision feedback system design. It is a C++ library with bindings for python, java, C# and etc, and can create application for Windows, OSX, Linux and UNIX on 32-bit and 64-bit architectures as well as mobile platforms. The source codes of wxWidgets can be complied as our need to various libraries, including the options such as static or shared, debug or release, Unicode, OpenGL, ODBC database and etc. Based on wxWidgets, the corresponding selections of Integrated Development Environment (IDE) and compiler are explicit: Code::Blocks (shown in Fig. 1) and Mingw. This combination of GUI, IDE and compiler are all free and can support many program languages.

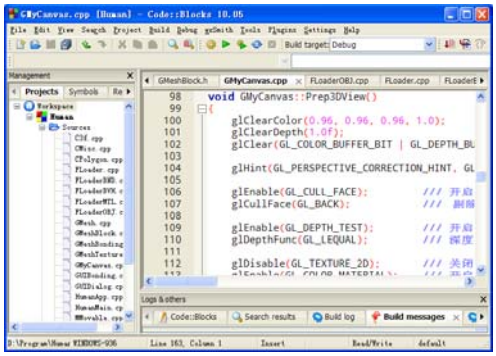


Fig. 1 The interface of Code::Blocks

Based on wxWidgets as GUI tool, OpenGL is the nature choice for graph interface programming. A special class wxGLCanvas has been developed in wxWidgets to support OpenGL programming. Modifying the compiler configures on wxGLCanvas and OpenGL is necessary in custom building. Load the OpenGL libraries as usual and the graph rendering codes are executed in a class inherited from wxGLCanvas.

In 3D human model, the graph texture is saved as a separate file. The texture files has different formats, so

loading and transforming them into bitmaps are very troublesome. A new piece of third-party software, FreeImage, is adopted here to help us make use of the texture file. Just like the above candidate tools, FreeImage is also free and open source. Not all of the graph format support transparent pattern and FreeImage can not automatically add transparent feature. Fortunately, a simple operation on bitmap arrays can transform color format from RGB to RGBA and the latter format supports transparent effect.

All the software selected is open source and free, and the details are shown as Table I.

TABLE I THE OPEN SOUSE FREE SOFTWARE

| The open source free software | Application |
|---|---|
| Mingw | Compiler |
| Code::blocks | Integrated Development Environment |
| wxWidgets | Graphical User Interface |
| wxFormBuilder | Rapid Application Development for wxWidgets |
| OpenGL | Graph rendering |
| FreeImage | popular graphics image formats |
| MakeHuman | modeling 3D humanoid characters |
| STL | Standard Template Library |

## III. 3D MODEL AND FILE LOADING

A special 3D model made by professional 3D software is much better than the graph directly drawn by OpenGL. The vivid human model can realize the immersive visualization effect. The commercial 3D software is abundant, but most of them are all-purpose and not good at human model. Fortunately, a piece of open source free software, Makehuman (shown in Fig. 2), specializes in human model making. Makehuman supports the easy modification of gender, age, tone, weight, height, ethnics and etc. The modeling results are exported as several formats: obj, mhx, dae, stl and etc. More libraries including clothes, eyes, skin and hair can be used to enhance the character details. But the clothes library has a shortcoming not being able to envelop the skins. Another piece of free software, blender can help to overcome this shortcoming with complicated modifications. We give up to adding clothes library and hope Makehuman workgroup can remedy this function defect as soon as possible.

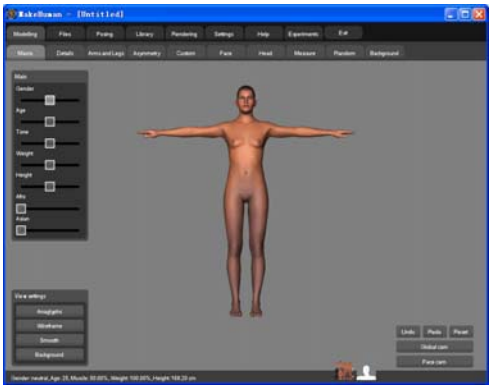

Fig. 2 The interface of Makehuman

The 3D file format has to be carefully selected to match the need of vision feedback system. Three selection standards are concluded as: skin rendering, skeleton moving and open source. The skin rendering is basic to 3D graph display, and a great number of polygons compose the mesh as human skin. The mesh not bonded with skeletons displays just as static graph, and can not follow the rehabilitation actions. So the skeleton information corresponding to each section of skins need to be saved in mesh file or as a separate file. The open source file format is convenient to file loader programming by ourselves not depending on commercial loader. Taking 3ds format as an example, it consists of mesh and skeleton information in one file, so it is often used to design vision feedback system. But 3ds format is not open source and the loading results are in pieces. Trying to load a locking format is undoubtedly harmful to the entire design of the vision feedback system.

Of the format supported by Makehuman, obj and bvh are mesh and skeleton file, respectively. An mtl file is exported as an appendant of the obj file and it contains the light, color and texture information for rendering. A graph file about texture is outputted as the supplement of mtl files, and the graph format may be jpg, bmp, png and etc. Under normal conditions, four files need to be loaded again in vision feedback system to recur a vivid character made in Makehuman. They are obj, bvh, mtl and png, respectively.

The obj, bvh and mtl files are ASCII format with special designing keywords, so loader programming is not difficult. A file stream class is defined and pointed to the file, and then a std::iterator from Standard Template Library (STL) is used to traverse file stream. The rest task of the loader is to store the stream contents in self-defined data structures.

The obj file mainly includes three kinds of definitions: vertex, face and group. The vertex has three types: the position of each vertex, the UV position of each texture coordinate vertex, normals. A list of vertices (at least three) and texture vertices define a polygon as a surface. The triangle is the common type. Noticeably, only the vertex index is used in surface definition for space saving. A surface set is called as a group which has the same materials.

An obj file is followed by one or more mtl files, from which one or more material descriptions are referenced by names. The mtl file defines the light reflecting properties for computer rendering. The ambient, diffuse and specular color of the material are declared as Ka, Kd and Ks, respectively. Color definitions are in RGB where each channel's value is between 0 and 1. The specular color is weighted using the specular coefficient Ns, and multiple illumination models are available per material. Textured materials are declared as a graph file name, and they belong the same properties as above.

Bvh is a type of motion capture file format used to import rotational joint data. A bvh file has two parts: a header section describing the hierarchy and initial pose of the skeleton; a data section containing the motion data. Only the header section is needed in vision feedback system, and our own data section will be imported to drive the joint. The header section begins with the keyword "HIERARCHY", then following the keyword "ROOT" and "JOINT". The bvh adopts a recursive

definition. Each segment of the hierarchy recursively defines its children. The first piece of information of a segment is the offset from its parent, which begins with the keyword "OFFSET".

A class is defined as the basis of the entire file format loaders. Then the inherit classes are used to take charge of the different formats. The STL container std::vector is the basic data structure saving the file contents. Some self-defined structures are used to transfer the useful information to the display and control program in the following.

## IV. BONDING THE SKIN AND SKELETON

The skin and skeleton data are independently loaded from different files. They are separately stored in isolated data structures without any connection. Bonding is in fact a process rebuilding the connection between skin and skeleton. The bonding process has two steps: matching and weighing. The matching is a step finding the map from skin to skeleton. Any piece of skin must be connected with at least one skeleton, or it can not be driven to follow the human move and action. The weighing is a step computing the weight between the skin and its mapping skeletons. The skin at joints is a typical example which is connected with two or more skeletons. So the membership degree to different skeletons needs to be described by a weight coefficient.

Sometimes the skeleton information is not available and the center lines are computed to make up for the absentation of skeleton information. Accidentally only a general skeleton is available without the appropriate scale and pose setting. The additional zooming and rotating operations is needed during matching process to carefully adjust skeletons. Fortunately, Makehuman has already appointed a special skeleton hierarchy for each 3D human model. In other words, the skeleton adjustment is finished by Makehuman and the rest task is just to determine the map relationship. The min-distance is the usual rule for automatic defining map, but it may fail in some human pose. For example, the skin at body sides may be mapped to upper limb skeletons when two arms clinging to the body. The bounding box is another bonding method. Its computation is simple but the bonding precision is not satisfying.

A bonding interface based on dialog frameworks is developed to replace the automatic bonding. The man-machine interaction through the bonding dialog realizes a type of manual bonding one by one. The bonding dialog includes three parts as shown in Fig. 3 : skeleton tree, skin database and map list. All the skeletons loaded from bvh file are displayed as a tree hierarchy that can be realized by the class wxTreeCtrl. Both the skin database and map list are in fact the lists inherited from the class wxListCtrl. The inserting and deleting operations of wxListCtrl depend on the component position. When multiple components are deleted at the same time, the position updating can not follow on time the deleting operation and often result in error deleting. A STL container, std::map is used as the background data structure for synchronizing the list updating. When the deleting command is sent, it is firstly executed in the std::map, and then the corresponding list is fully refreshed by the background contents.

All the skin blocks not being bonded is temporally stored in the skin database. The user can switch the map list by double clicking the tree node. When the skin block is selected and inserted into the map list, it has been bonded to the current skeleton. The inner data structure of the current skeleton synchronously updates the bonding information. If the bonding information needs to be modified, the skins can be deleted from the map list to release the bonding.

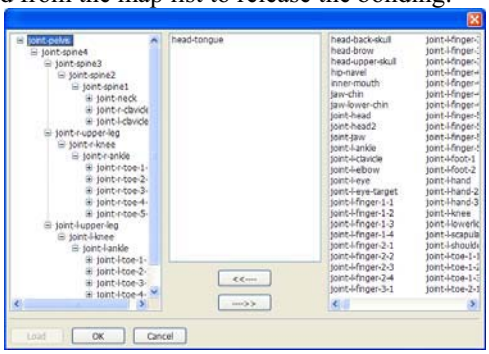


Fig. 3 The bonding interface

The skeleton hierarchy is displayed as matchstick shape. The skeleton is on the center axis of the skin surface and is covered by skins when natural displaying. The skins are set to be transparent and help us to clarify the map relationship. Multiple display modes are set as Fig. 4 to distinguish the bonding stages. All the skins are in grey color by default; the skin selected is in highlight yellow color and the skin bonded is in texture mode. The model loaded is three-dimension so some skins can not been seen from the front, and it can be freely rotated by mouse grasping to show the back skin. When all the skins are scattered one by one into the map list, the bonding task is finished and the bonding result is saved as a text file. The bonding information will be loaded again during loading the human model.

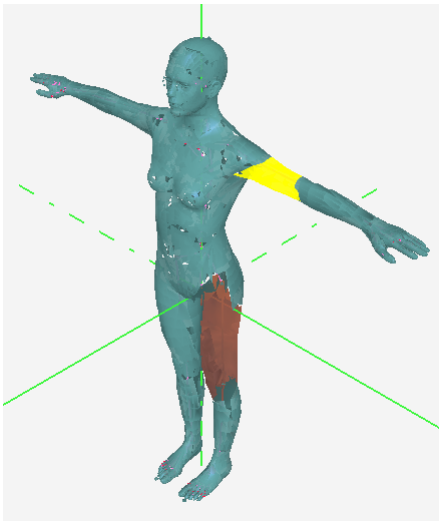

Fig. 4 The coloring style for bonding process

The principle of bonding algorithm can be described as the following equation:

$$v^{new} = \sum_{i=1}^{N} w_i \mathbf{M}_i^{new} \left( \mathbf{M}_i^{old} \right)^{-1} v^{old} \ . \quad (1)$$

$v$ is the coordination of the skin vertex; $\mathbf{M}$ is the transform matrix of the skeleton; $w$ is the weight and $i$ is the number of the skeleton. The superscripts *old* and *new* represent the original and current positions, respectively.

There are two bonding weight: rigid and flexible. The rigid bonding means that the skin is bonded with the sole skeleton and the weight is set as 1. The rigid weight is rough and may result in the skin distorting. The flexible weight is more reasonable as well as more complex. There are many expert weighting computation algorithms in commercial software. Here only line bonding algorithm, template fitting, is introduced.

Baran and *et al* [21] present a new weighting computation: simulating template fitting. The human model is considered as a thermal conductor. The template of heating skeleton $i$ is 1, and the template of the others is 0. After fitting, the template at each vertex is set as the weight originated from skeleton $i$. The weighting computation equation is

$$-\Delta w^i + \mathbf{H} u^i = w^i \ . \quad (2)$$

Where $\Delta$ is Laplace operator; $u$ is a vector and $u_j^i = 1$ if the skeleton $i$ is the closest to vertex $j$, or $u_j^i = 0$. $\mathbf{H}$ is a diagonal matrix and $H_{jj}$ is the template weight from vertex $j$ to the closest skeleton. Usually $H_{jj} = c \div w_{ji}^2$ and the const $c$ is set as 0.105~0.385. When the linear weight is added, the skin deformation at joints will be more realistic.

## V. DATA-DRIVEN MOTION RENDERING

After all the 3D model files are successfully loaded, the information has been preserved in data structures, and the skins have been bonded with appropriate weights. The left task is to render the human model on the screen. A panel is created in the man-machine interface as wxGLCanvas's parent window. When the wxGLCanvas instance is constructed, a function SetCurrent is called to set the instance as the context environment for rendering. Then all kinds of OpenGL commands are called to set up the implicit state for rendering context.

OpenGL provides three transformation functions: the translating function glTranslate; the rotating function glRotate and the scale zooming function glScale. OpenGL provides four transformation types: the viewing transformation; the modeling transformation; the projection transformation and the viewport transformation. Before each transformation is implemented, the function glLoadIdentity must be called to set the current matrix as the identity one. By these transformation types, the 3D model can be displayed as a 2D graph in a screen.

Before beginning the graph rendering, many environment settings are needed to initialize. The lighter is turned on and its attributes are set based on what the mtl file defines. The color and depth buffers are cleared by the prior setting value. The backface culling and depth testing are enabled to save rendering time consumption. The 2D texture and colorful material are switched on to support the color rendering. The color blending is enabled to realize the transparent effect.

Drawing a polygon with OpenGL is easy. One vertex is rendered from its normal, texture coordination to its position coordination. All the vertexes of the polygon are rendered in turn by counter-clockwise. The surface surrounded by the polygon is automatically rendered as long as all the vertexes are rendered. The polygon color should be defined before rendering. Because OpenGL is a type of language based on state machines, the settings keep being effect unless they are modified. As same as the color setting, the texture settings needed to be loaded only once, too. Multiple texture loadings are redundant and may result in graph buffers breaking down.

The transparent effect is a little trouble. The color format must be changed by manual from RGB to RGBA because FreeImage has not the related function. The alpha value is inserted into the color array one by one. The rendering order is also important: the first is opaque objects; then the depth buffer is set as read-only mode; finally the transparent objects are rendered, and the depth buffer resume to the read-write mode. Realizing the transparent effect is critical to skin bonding, because the skeleton is visible only when the skin is rendered as transparent mode.

All the OpenGL commands for rendering can be temporarily saved as a display list for the following calling. Each display list has a sole number automatically assigned by OpenGL. As long as the list number is appointed during the display refreshing, the corresponding command codes will be executed in term of the saving. If the rendering need to be modified, a new list number is assigned after the old number and the related codes are destroyed. In other words, the rendering process is separated into two parts: the background codes preparing and the foreground display refreshing. The separate operation mode makes the motion rendering to be very easy. A buffer queue is defined to saving the display list numbers, each of which corresponds to a section of codes for motion updating. The timing samples of motion information are acquired to generate new display lists which are saved in buffer queue. The list number at the top of queue is pushed up to execute while screen updating.

The multi-thread technique is prevalent in GUI program. The background sampling and GUI displaying are separated into two different threads to ensure the real-time performance as possible as it can. A difficult problem is the data transferring from the background thread to GUI thread. The data interlocking is the general sharing mechanism, but it is inefficient to frequently locking and unlocking the shared data. A new data transferring mechanism based on self-defined events is designed in the vision feedback system. wxWidgets supports us to define our own events inherited from the class wxCommandEvent or wxNotifyEvent. Different to the common class definition, the event type list and the event map macro must be declared and defined. When

the new samples are obtained, they are inserted into an instance of the self-defined event and sent to the aim class by calling the function wxPostEvent. On the other hand, the aim class needs to define the event table including the response function for the self-defined event.

In conclusion, the workflow of motion rendering can be expressed as following: the data sampled by motion sensors are loaded in the self-defined event; the event is transferred to GUI thread; the data carried on the event are used to generate the display list; the list number is called to update the display. As long as the motion sensor samples are continually updated, the human model will show continuous motion. A primary framework is shown as Fig. 5.

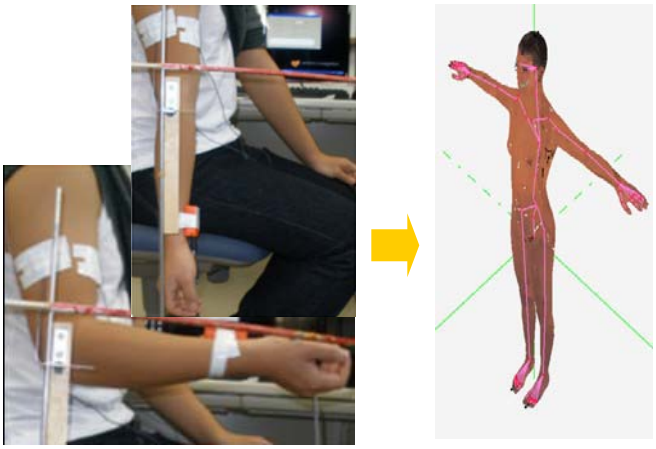

Fig. 5 The motion rendering driven by motion data

## VI Conclusions

The rehabilitation paradigm AAN takes the patient's voluntary efforts into account rather than predefining moving trajectory. A vision feedback system is developed in this study to inspire patients' activeness and correct their intentions during rehabilitation training in term of AAN paradigm. All the third-party software used in the system is free and open source, from IDE, GUI, human character to graph rendering. This mini system can be equipped with portable rehabilitation robots to decrease the training cost, or used to build more complex VR environments for rehabilitation. In the future a new multi-joint motion data sampling system will be developed as the hardware supplement to free from MTx sensors.

## Acknowledgment

This study was supported by Kagawa University Characteristic Prior Research fund 2011, and partially supported by National Natural Science Foundation of China (Grant No. 61101022 and 61105087).

## References

[1] H. S. Lo and S. Q Xie, "Exoskeleton robots for upper-limb rehabilitation: State of the art and future prospects," *Medical Engineering & Physics*, vol. 34, no. 3, pp. 261-268, April 2012.

[2] R. C. V. Loureiro, W. S. Harwin, K. Nagai and M. Johnson, "Advances in upper limb stroke rehabilitation: a technology push," *Medical & Biological Engineering & Computing*, vol. 49, no. 10, pp. 1103-1118, October 2011.

[3] A. C. Lo, P. D. Guarino, L. G. Richards and et al, "Robot-assisted therapy for long-term upper-limb impairment after stroke in clinic," *The New England Journal of Medicine*, vol. 362, no. 19, pp. 1772-1783, May 2010.

[4] P. S. Lum, C. G. Burgar and P. C. Shor, "Evidence for improved muscle activation patterns after retraining of reaching movements with the MIME robotic system in subjects with post-stroke hemiparesis," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 12, no. 2, pp. 186-194, June 2004.

[5] Z. B. Song and S. X. Guo, "Development of a novel exoskeleton rehabilitation devices and implementation of bilateral upper limb motor movement," *Journal of Medical and Biological Engineering*, vol. 32, no. 3, 2012, in press.

[6] M. Mihelj, T. Nef and R. Riener, "A novel paradigm for patient-cooperative control of upper-limb rehabilitation robots," *Advanced Robotics*, vol. 21, no. 8, pp. 843-867, (2007)

[7] L. L. Cai, A. J. Fong, C. K. Otoshi and et al, "Implications of assist-as-needed robotic step training after a complete spinal cord injury on intrinsic strategies of motor learning variability," *The Journal of Neuroscience*, vol. 26, no. 41, pp. 10564-10568, October 2006.

[8] H. Sveistrup, "Motor rehabilitation using virtual reality," *Journal of NeuroEngineering and Rehabilitation*, vol. 1, no. 1, pp. 1-10, December 2004.

[9] A. R. Samia and S. Afaf, "Virtual reality use in motor rehabilitation of neurological disorders: A systematic review," *Middle-East Journal of Scientific Research*, vol. 7, no. 1, pp. 63-70, 2011.

[10] S. K. Ong, Y. Shen, J. Zhang and A.Y.C. Nee, "Augmented reality in assistive technology and rehabilitation engineering," *Handbook of Augmented Reality*, vol. 2, pp. 603-630, 2011.

[11] M. Trlep, M. Mihelj, U. Puh and M. Munih, "Rehabilitation robot with patient-cooperative control for bimanual training of hemiparetic subjects," *Advanced Robotics*, vol. 25, no. 15, pp. 1949-1968, 2011.

[12] E. C. Liu, J. F. Sui, Y. Christopher and L. H. Ji, "Double visual feedback in the rehabilitation of upper limb," *Lecture Notes in Computer Science*, vol. 6768, pp. 384-388, 2011.

[13] R. Secoli, M. H Milot, G. Rosati and D. J. Reinkensmeyer, "Effect of visual distraction and auditory feedback on patient effort during robot-assisted movement training after stroke," *Journal of NeuroEngineering and Rehabilitation*, vol. 8, no. 1, pp. 1-10. April 2011.

[14] Y. Q. Tao, H. S. Hu and H. Y. Zhou, "Integration of vision and inertial sensors for home-based rehabilitation," in: *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 18 2005.

[15] B. R. Brewer, M. Fagan, R. L. Klatzky, and Y. Matsuoka, "Perceptual limits for a robotic rehabilitation environment using visual feedback distortion," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, no. 1, pp. 1-11, March 2005.

[16] H. P. Wu, J. H. Liu, H. Handroos and et al, "Virtual reality based robotic therapy for stroke rehabilitation an initial study," in: *IEEE International Conference on Mechatronics and Automation*, Beijing, China, August 7-10 2011.

[17] C. Loconsole, R. Bartalucci, A. Frisoli and M. Bergamasco. "An online trajectory planning method for visually guided assisted reaching through a rehabilitation robot," in: *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9-13, 2011.

[18] E. Vergaro, M. Casadio, V. Squeri and P. Giannoni, "Self-adaptive robot training of stroke survivors for continuous tracking movements," *Journal of NeuroEngineering and Rehabilitation*, vol. 7, no. 13, pp. 1-12, March 2010.

[19] A. Duschau-Wicke, J. von Zitzewitz, A. Caprez and et al, "Path control: A method for patient-cooperative robot-aided gait rehabilitation," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 18, no. 1, pp. 38-48, February 2010.

[20] F. Wang, D. E. Barkana, N. Sarkar, "Impact of visual error augmentation when integrated with assist-as-needed training method in robot-assisted rehabilitation," vol. 18, no. 5, pp. 571-579, October 2010.

[21] I. Baran, J. Popovic, "Automatic rigging and animation of 3d characters," ACM Transaction on Graphics, vol. 26, no. 3, pp. 1-8, July 2007.