Chapter 6
Human-Centered 3D Home Applications via Low-Cost RGBD Cameras

Zhenbao Liu, Shuhui Bu and Junwei Han

Abstract In this chapter, we will introduce three human-centered home 3D applications realized by virtue of low-cost RGBD cameras. The first application is personalized avatar for user via multiple Kinects, which can reconstruct a real human and provide personalized avatars for everyday users and enhance interactive experience in game and virtual reality environments. The second application automatically evaluates energy consumption of users in gaming scenarios by a model with tracked skeleton, which may help users to know their exercise effects and even diet or reduce their weights. The final application presents a real-time system that automatically classifies the human action acquired by consumer-priced RGBD sensor.

6.1 Personalized Avatar for User via Multiple Kinects

In traditional human-centered games and virtual reality applications, skeleton of human is commonly captured via consumer-priced cameras or professional motion capture devices to animate an avatar to follow his movements. In this section, we

Z. Liu and S. Bu were supported by NSFC (61003137, 61202185), Fundamental Research Funds for the Central Universities(310201401JCQ01012, 310201401JCQ01009), Shaanxi NSF(2012JQ8037), and Open Fund from State Key Lab of CAD&CG of Zhejiang University. J. Han was supported by the NSFC under Grant 61005018 and 91120005, NPU-FFR-JC20120237 and Program for New Century Excellent Talents in University under grant NCET-10-0079.

Z. Liu · S. Bu (✉) · J. Han
Northwestern Polytechnical University, Xi'an, China
e-mail: bushuhui@nwpu.edu.cn

Z. Liu
e-mail: liuzhenbao@nwpu.edu.cn

J. Han
e-mail: jhan@nwpu.edu.cn

L. Shao et al. (eds.), *Computer Vision and Machine Learning with RGB-D Sensors*,
Advances in Computer Vision and Pattern Recognition,
DOI: 10.1007/978-3-319-08651-4_6

propose a novel application that automatically reconstructs a real 3D moving human captured from multiple Kinect RGBD cameras in the form of polygonal mesh, which may help users to really enter a virtual environment or even collaborative immersive environments. Compared to 3D point cloud, 3D polygonal mesh is commonly adopted to represent objects or characters in games and virtual reality applications. The vivid 3D human mesh can enormously promote the immersion when he interacts with a computer. The proposed method includes three key steps to dynamically realize 3D human reconstruction from noisy RGB image and depth data captured in a distant distance. We first recover 3D scene represented in point cloud from scanned RGBD data. We then filter noisy and unorganized point cloud and obtain a relatively clean 3D human point cloud. A complete 3D human mesh is reconstructed from the filtered point cloud using Delaunay triangulation and Poisson surface reconstruction. A group of experiments demonstrates the reconstructed 3D human meshes, and these dynamic meshes with different poses are placed in a virtual environment. It could be used to provide personalized avatars for everyday users and enhance interactive experience in game and virtual reality environments.

6.1.1 Introduction

Real 3D scene and human reconstruction from a professional 3D scanner have been well studied in multimedia, virtual reality, and computer graphics [1]. It has generated satisfactory results adaptable to industrial inverse engineering and production design based on virtual reality. However, the technique cannot be directly applied in home-centered amusements because of high price, large volume, difficult operation, and computational burden. In recent years, portable RGBD cameras with low cost and easy operation, for example, Kinect [2], is highly appealing and becoming more widespread. Han et al. [3] investigate recent Kinect-based computer vision algorithms and applications and classify these algorithms according to the type of vision problems. These topics include preprocessing, object tracking and recognition, human activity analysis, hand gesture analysis, and indoor 3D mapping. However, the type of cameras generates low-quality color and depth images, which becomes a main constraint on providing fully immersive and virtual applications. Many researchers have payed close attention to recent developments on high-quality scene and human generation.

There have been several pioneering works focusing on interesting 3D virtual applications of RGBD sensors. Alexiadis et al. [4] build a real-time automatic system of dance performance evaluation via Kinect RGBD sensor and provide visual feedback to beginners in a 3D virtual scene. Bleiweiss et al. [5] propose a solution to animating in-game avatars using real-time motion capture data and blend actual movements of players with pre-defined animation sequences. Pedersoli et al. [6] provide a framework for Kinect enabling more natural and intuitive hand gesture communication between human and computer devices. Tong et al. [7] present a novel scanning

system for capturing different parts of a human body at a close distance, and then reconstructing 3D full human body.

Ye et al. [8] propose an algorithm for creating free-viewpoint video of interacting humans using three handheld Kinect cameras. They reconstruct deforming surface geometry and temporal varying texture of humans through estimation of human poses and camera poses for every time step of the RGBD video. Barmpoutis [9] reconstruct 3D model of the human body from a sequence of RGB-D frames by means of parameterization of cylindrical-type objects using Cartesian tensor and b-spline bases along the radial and longitudinal dimension. The reconstruction is performed in real time, while the human subject moves arbitrarily in front of the camera. The tensor body is fitted to the input data using segmentation of different body regions, robust filtering, and energy-based optimization.

Differently from the above applications, we attempt to solve the problem of dynamic 3D human reconstruction from data acquired by means of multiple RGBD sensors located in a distant distance. This is a similar scene setting as home game. Nevertheless, several major difficulties must be faced, which include (1) RGB image and depth data simultaneously sampled from the same RGBD sensor in a low resolution is difficult to be calibrated and coupled to 3D textured point cloud without distortion and (2) the coupled point cloud is too sparse and noisy to reconstruct a detailed and smooth mesh. In order to overcome these problems, we first propose several 3D human filters to extract relatively pure point cloud of human from a whole 3D scene. An initial triangulation with holes and burrs is then reconstructed based on the point cloud. We finally fill all the holes and smooth the surface via poisson surface reconstruction. The reconstructed 3D human is dynamically placed into a virtual environment and able to follow the movements of users. See Fig. 6.1 for an illustration.

The section is organized as follows. The colored point cloud of scene is first generated in Sect. 6.1.2. Section 6.1.3 illustrates how we filter the noisy and unorganized 3D scene point cloud to get a relatively clean human point cloud. We introduce two steps to convert the point cloud to 3D mesh in Sect. 6.1.4. Section 6.1.5 demonstrates personalized avatar of user in a virtual scene.

6.1.2 Point Cloud Generation

We adopt the dataset captured by five Microsoft Kinects and their registration parameters, which are provided by 3DLife/Huawei Grand Challenge of ACM Multimedia 2013 [10]. The dataset contains five groups of Kinect data recording the same scene. The five Kinect sensors are, respectively, placed in different positions and different directions. Each Kinect outputs a sequence of depth and color images, which are encoded with video format. Each one of the five captured videos only records a part view of the whole 3D scene due to the limited visual angle of Kinect camera. The raw depth map and its corresponding color map of each view are separately extracted by means of OpenNI develop toolkit. Both depth and color images have the same

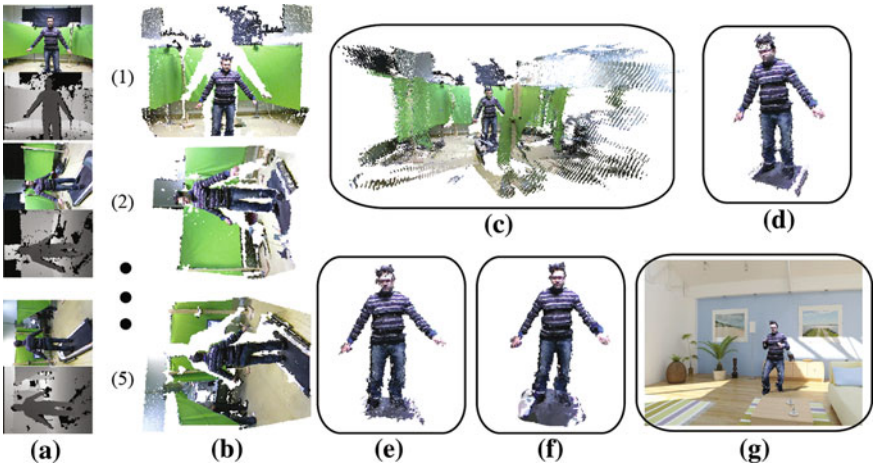

Fig. 6.1 Overview of the steps in our personalized avatar implementation. **a** RGB and depth images from (*1*) to (*5*) captured from five Kinects. **b** Five respective RGB-D calibrated images. **c** Registered point cloud of scene. **d** Filtered 3D human point cloud. **e** Delaunay triangulation. **f** Poisson mesh reconstruction. **g** Movement of personalized avatar in a virtual scene

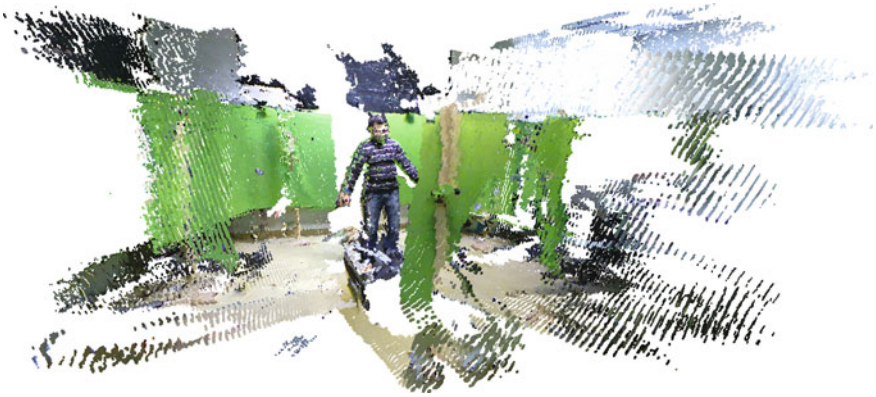

Fig. 6.2 Registered whole scene in the form of 3D point cloud. We first calibrate the depth and color map using a recent algorithm [11] for each Kinect data and map the color map to the depth map using camera parameters to get five colored point clouds. The five point clouds are then registered into one whole 3D scene

resolution with 480 × 640 pixels. Because depth and color images come from different cameras on the same Kinect and are not completely overlapped, we adopt a recent algorithm [11] to calibrate them in advance. The calibration model does not use depth discontinuities in the depth image and accordingly is flexible and robust to noise. We use the calibrated camera parameters to recover the five 3D partial views, and then register the five 3D partial views into one whole 3D point cloud scene. The reconstructed whole 3D scene in the form of point cloud is shown in Fig. 6.2.

6.1.3 3D Human Filters

Although the color and depth images have been calibrated and registered into 3D point cloud without distortion, it can be seen that the point cloud is still very noisy and mixed with a lot of isolated points and wrongly colored points. This leads to the difficulty in reconstructing the point cloud to a clean polygonal mesh. Moreover, many details representing realistic effects of a user, for example, face and clothes, should be recovered in a high-quality way.

In order to attain the above objective, we first have to extract a whole human from the 3D scene point cloud. Because many noisy points and background points have been mixed into the point cloud of human in the step of scene generation, we consider how to remove these redundant points and extract a relatively clean point cloud of human from the whole 3D scene. We propose three simple and efficient filters to finish the task, which are band-pass filter $F(b)$, background color filter $F(c)$, and distance filter $F(d)$.

We first use a band-pass filter $F(b)$ to get a coarse human point cloud. It is expressed as the following form.

$$F(b) * p_i = \begin{cases} p_i, & x \in [x_0, x_1], y \in [y_0, y_1], z \in [z_0, z_1] \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

where $\{[x_0, x_1], [y_0, y_1], [z_0, z_1]\}$ denotes a bounding box of the human point cloud. p_i is a point in the point cloud. $F(b) * p_i$ means applying the filter to the point p_i.

Here, the bounding box is built by virtue of a recent high-level skeleton tracking algorithm [15]. The algorithm designs an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Its classifier is trained by efficient randomized decision forests, considering a large number of characters from short to tall and thin to fat, many poses, models with different hair styles and clothing. Once we get the positions of the skeleton joints tracked, the bounding box of the whole skeleton is simply computed by utilizing the maximum and minimum values of all the joint coordinates. We empirically enlarge the bounding box of skeleton by 10 filter $F(b)$. The filter passes a coarse human point cloud as shown in Fig. 6.3a and reject outside points. We see the point cloud is still disturbed by background noises. In order to keep a better point cloud for next mesh reconstruction, we refine the boundary of coarse human point cloud. A background color filter $F(c)$ is considered to get rid of the noisy background color points on the boundary. Finally, we use another distance filter $F(d)$ to delete the isolated points which may cause wrong faces in mesh reconstruction.

$$F(d) * p_i = \begin{cases} p_i, & d(p_i) < d_0 \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

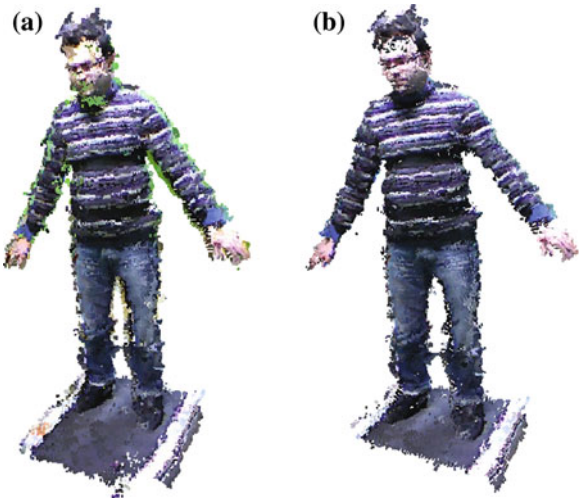

Fig. 6.3 **a** Coarse human point cloud. **b** Final human point cloud. We first use a band-pass filter to get the coarse human point cloud (**a**). By applying other two filters, background color filter and distance filter, the final point cloud (**b**) is obtained

$d(p_i)$ is the distance between p_i and its nearest point, and we practically set the constant threshold value d_0 to 10. After passing these three filters, a relatively fine scattered point cloud is generated for next mesh reconstruction, as shown in Fig. 6.3b.

6.1.4 Human Mesh Reconstruction

The human point cloud with color generated above has three defective problems, which may cause incorrect mesh reconstruction. We first solve the organization problem of these unordered 3D points and adopt polygonal form to topologically connect these points. The second problem to be addressed is that there are many holes in the point cloud because of low image resolution of Kinect, placement of Kinect in a distant distance, and missed points while scanning human in different directions. Another problem we find is that the point cloud is not smooth and contains a lot of burrs on the boundary.

Therefore, we attempt to introduce a two-stage solution to these low-quality problems. In the first stage of mesh reconstruction, we preliminarily triangulate the unorganized point cloud and make these massive points connect with spatial neighbors. We solve the problem of holes and burrs in the second stage, and it can be cast as an issue of Poisson surface reconstruction. Because global Poisson equation considers all the points simultaneously, it is considered highly robust to small data noises.

During the first stage, we topologically connect and triangulate the unorganized point cloud by means of Delaunay triangulation [12] to get a coarse human mesh from

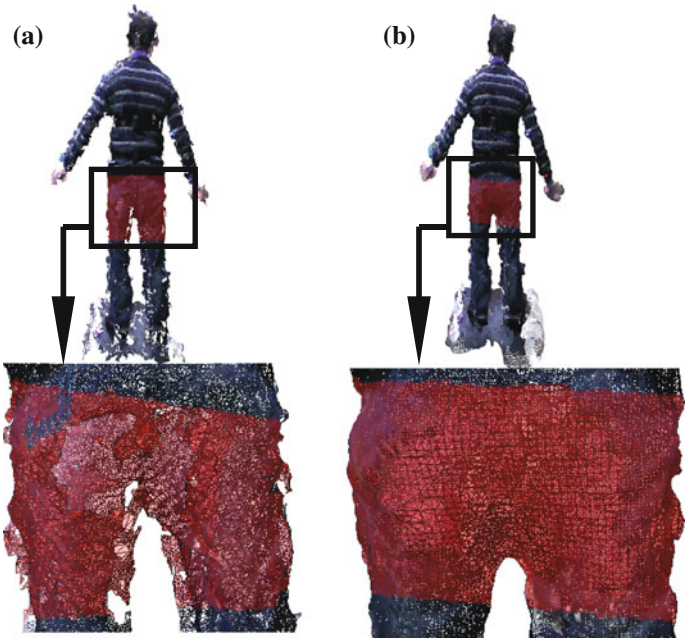

Fig. 6.4 **a** Local amplification in Delaunay triangulation. **b** Local amplification in Poisson surface reconstruction. It can be seen the surface is smoothed after introducing Poisson reconstruction

the scattered point cloud. In order to obtain an adaptive human mesh, it is feasible to adjust the density of triangles according to the amount of points in different parts of human. However, we find that Delaunay triangulation is susceptible to undersampling and outliers. For example, there are some holes and burrs in the triangulated human mesh shown in Fig. 6.4a.

In order to overcome the problem, in the second step, we adopt the Poisson mesh reconstruction [13] to refine the initial mesh, fill the holes on the surface, and remesh it. The surface reconstruction is seen as the solution to a Poisson equation of isosurface as follows.

$$\Delta\phi = \nabla \cdot \mathbf{V}, \tag{6.3}$$

where ϕ is an indicator function and Δ denotes its Laplace operator. The divergence ∇ of a vector field $\mathbf{V}$ equals Laplacian of the scalar function. The above Poisson equation is actually equivalent to expected approximation and reconstruction:

$$\phi = \operatorname{argmin} \|\nabla\phi - \mathbf{V}\|, \tag{6.4}$$

which means the gradient of the implicit function ϕ of an input point cloud $\{p_i\}$ fits the vector field based on these points and their normal vectors $\{\mathbf{v}_i\}$. The form of the vector field is discretized in a subdivision octree as follows.

$$\mathbf{V} = \sum_{p_i} \sum_{j \in n(p_i)} \alpha_{ij} b_j(p_i) \mathbf{v}_i, \tag{6.5}$$

where $n(p_i)$ denotes the set of the eight closest octree nodes of each point p_i and j is the node index. α_{ij} is the interpolation weight. $b_j(p_i)$ is a transformed function on each octree node n_j. It is worth noting that it is similar as wavelet built on the 3D regular grid. More specifically, each transformed function is obtained by translating and scaling a fixed basis function

$$b_j(p_i) = b\left(\frac{p_i - c(n_j)}{w(n_j)}\right) \frac{1}{w(n_j)^3}, \tag{6.6}$$

where $b(p_i)$ denotes the fixed basis. $c(n_j)$ is the center of the node n_j and $w(n_j)$ is its width. The basis function has a uniform form, that is, the convolution with a box filter $f(x)$ (or $f(y)$, $f(z)$) t times separately on each dimensional coordinate $\{x, y, z\}$. The convolution can be expressed as

$$b(x, y, z) = (f(x) * f(x))^t (f(y) * f(y))^t (f(z) * f(z))^t, \tag{6.7}$$

where each box filter, for example, $f(x)$, has the following form

$$f(x) = \begin{cases} 1, & |x| < 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{6.8}$$

The approximation problem in Eq. 6.4 can be converted by projecting it onto the space spanned by bases b_j, $j \in [1, |T|]$. $|T|$ is the number of octree nodes. The solution to ϕ is equivalent to minimizing

$$\begin{aligned} &\sum_j \left\| \langle \Delta\phi - \nabla \cdot \mathbf{V}, b_j \rangle \right\|^2 \\ &= \sum_j \left\| \langle \Delta\phi, b_j \rangle - \langle \nabla \cdot \mathbf{V}, b_j \rangle \right\|^2, \end{aligned} \tag{6.9}$$

which can be written in a matrix form, and the Laplace matrix L with $|T| \times |T|$ elements is defined so that $L\phi$ returns the dot product of the Laplacian with each base b_j. Each element of the sparse and symmetric matrix has the following expression

$$L_{jk} = \left\langle \frac{\partial^2 b_j}{\partial x^2}, b_k \right\rangle + \left\langle \frac{\partial^2 b_j}{\partial y^2}, b_k \right\rangle + \left\langle \frac{\partial^2 b_j}{\partial z^2}, b_k \right\rangle. \tag{6.10}$$

Then, the Poisson equation reduces to a well-conditioned sparse linear system

$$L\phi = \mathbf{v}, \tag{6.11}$$

where $\mathbf{v}$ is a $|T|$-dimensional vector whose element is

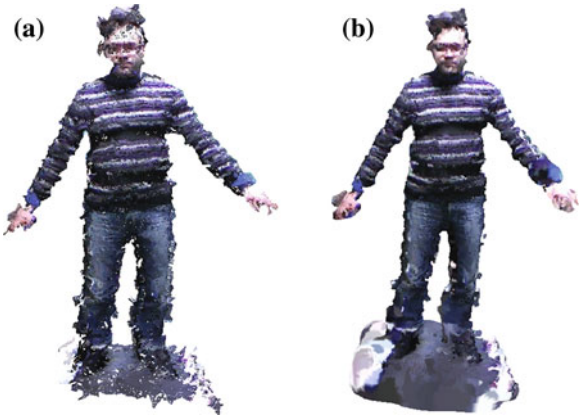

Fig. 6.5 **a** Delaunay triangulation. **b** Poisson surface reconstruction. Delaunay mesh is an initial result only by triangulating the human point cloud. It can be seen that the mesh contains a lot of holes and burrs. Poisson surface is then approximated from the Delaunay triangles as a way to refine the initial mesh. These holes and burrs are filled and smoothed, respectively, in the remeshing result

$$v_j = \langle \nabla \cdot \mathbf{V}, b_j \rangle . \tag{6.12}$$

The above linear system is solved using a conjugate gradient solver. After obtaining the indicator function ϕ, Marching Cubes [14] based on octree representations are adopted to extract the iso-surface and build a 3D human mesh. The solution can create a smooth surface that robustly approximate noisy points, as illustrated in Fig. 6.4b. The reconstruction comparison between the initial Delaunay triangulation and Poisson surface reconstruction is shown in Fig. 6.5. In the parameter settings, the maximum depth of octree is empirically set to be eight for achieving balance between reconstruction quality and run time.

Since Poisson surface reconstruction does not allow the color information attached to the point cloud, which is very important for recovering a realistic human, we regain the texture of the reconstructed mesh by searching the correspondence between points before reconstruction and vertices on the reconstructed surface. An efficient kd-tree algorithm is designed to realize this task. Finally, we obtain a relatively smooth textured mesh without holes, as illustrated in Fig. 6.5b. See Fig. 6.6 for more reconstruction results.

6.1.5 Experimental Results

We implemented the proposed dynamic personalized avatar algorithm from noisy RGBD data based on C++, OpenNI [17], and OpenCV [16]. OpenNI packages are first employed to obtain the raw depth and color maps from the provided dataset. By slightly modifying the code provided by OpenNI develop toolkit, we compute the

Fig. 6.6 The reconstruction examples of three users with different poses

RGB value of each pixel in the color map and the distance of each pixel in depth map to Kinect. Because there exist offsets between color values and depth values, we align the depth and color maps with the aid of a recent calibration method [11]. And then, five camera images are registered to be a large 3D scene in the form of point cloud in OpenCV. 3D human filters are introduced to obtain a relatively clean 3D human point cloud, and we reconstruct its 3D mesh via Delaunay triangulation and Poisson surface reconstruction. The averaged time for reconstruction of human is 0.47 seconds on a modern computer with i7 CPU with 16G memory, and the implementation can be seen as a near real-time solution to personalized avatar for user via multiple Kinects. It should be noted that in practical applications, the reconstruction of realistic 3D human mesh can be performed only once, for example, in the stage of system initialization.

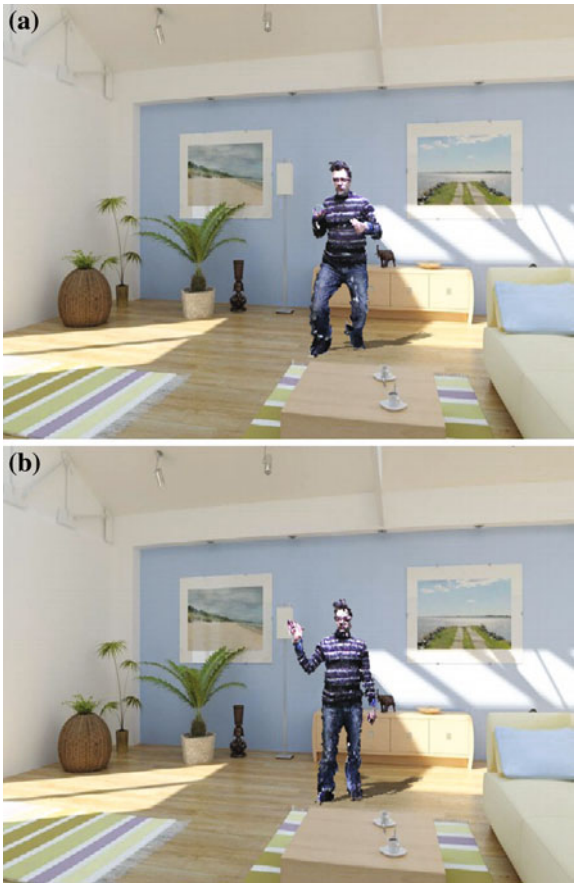

Fig. 6.7 **a** Reconstructed boxing action. **b** Reconstructed waving action. The reconstructed realistic 3D human is placed in a virtual scene and follows user movements

The movement of skeleton can be mapped onto the preliminary 3D human in real time.

In order to further demonstrate the realistic effect of personalized avatar, we dynamically reconstruct a 3D human mesh for each frame in these videos and visualize the sequential results in a gaming development software, Unity [18]. We reconstructed a sequence of moving 3D human meshes with different postures to test our method, as shown in Fig. 6.7. These 3D human meshes are dynamically imported into a virtual scene built in Unity and compose a realistic and meaningful action which approximates the actual human movements in the scene of game or virtual reality. It will enhance the interactive experience of users in practical applications.

6.1.6 Conclusion

In this section, we discussed the novel proposal about personalized avatar for user via multiple Kinects. The core of the implementation process lies in how to reconstruct a realistic human from noisy RGB data. The process is composed of three key steps, that is, point cloud generation, 3D human filters, and human mesh reconstruction. This novel application could help a user to see himself in a virtual scene using low-cost RGB-D cameras. This will increase the interactive experience with the computer in a virtual world. In the future, we will concentrate on continuing to improve the quality of reconstructed human mesh from sparse RGB-D data in a noisy environment. Neighbor frames will be considered as complementary information and several reconstructed human in a non-rigid form will be registered to a more complete 3D mesh. We also try to realize its practical applications such as virtual fitting room for validating its performance further.

6.2 Evaluating User's Energy Consumption Using Kinect-Based Skeleton Tracking

In this section, we propose a refreshing application that automatically evaluates player's energy consumption in gaming scenarios by a model with tracked skeleton, which may help users to know their exercise effects and even diet or reduce their weights. We develop a program to compute the energy consumption in real time by analyzing data captured from Microsoft Kinect and also give a cue in the dynamic interaction. We model 3D human skeleton by joining different body parts with 15 nodes and decompose player action into rigid body motions of these parts. Amount of energy consumed in the action is calculated as the sum of powers required to overcome gravity of each part. Experimental results show that instantaneous and total energy consumption of different dancers can be stably calculated. The hardware system is based on low-price Kinect and easily accepted by users. The proposed application also provides a quantitative approach which help users to control their dining and exercise intensity.

6.2.1 Introduction

Home-oriented virtual reality technologies become increasingly important in real-time realistic interaction games. They bring players rich experience by placing players in virtual environments. As a new kind of devices based on infrared structured light, depth sensors with low price such as Microsoft Kinect [2] have attracted much attention among not only game users but also researchers and developers. It is possible to use depth sensors like Microsoft Kinect to capture human motion in an easy and robust way. This will help many interactive games to animate an avatar and promote user experience in games.

Some researchers have begun to use depth sensor data to construct 3D virtual applications. Tong et al. [7] present a novel scanning system for accurately capturing 3D full human body model by using multiple Kinects, which could be used to provide personalized avatars for everyday users. Bleiweiss et al. [5] blend player's actual movements tracked using a depth sensor with pre-defined animation sequences. They aim at visually enhancing the player's motion to display exaggerated and supernatural motions. Suma et al. [20] provide us a middleware to facilitate integration of full-body control with virtual reality applications and video games using OpenNI-compliant depth sensors. Alexiadis et al. [4] propose an interesting application and evaluate dance performances of students against a gold-standard performance and provide visual feedback to the student dancer in a 3D virtual environment.

Different from above applications, we propose a novel system that automatically evaluates player's energy consumption in gaming scenarios, which may help users to know their exercise effects and even diet or reduce their weights. We attempt to solve the problem of real-time computation of energy consumption while dancing and also give a numerical feedback in the interaction environment. We model 3D human skeleton by connecting different body parts using 15 joints and decompose player's action into rigid body motions of these parts. Amount of energy consumed in the action is calculated as the sum of powers required to overcome gravitational potential energy. The dancing action is seen as the process of burning calories or fat. We stably obtain quantitative energy consumption of each dancer and convert it to the value of burned fat. Players could interactively know the exercise effects during dance.

The section is organized as follows. Section 6.2.2 describes how to track the skeleton of Kinect captured data. We provide a energy consumption model by introducing power of rigid motion in Sect. 6.2.3. We demonstrate the capability of stably computing energy consumption of users by a group of experiments in Sect. 6.2.4 and allow players to interactively know their exercise effects.

6.2.2 Kinect Skeleton Tracking

We use the provided dancer dataset captured using a Microsoft Kinect in 3DLife/Huawei Challenge of ACM Multimedia. We take 10 dancers' videos from the set of dance videos. Each video is composed of a sequence of depth images, which are encoded with video format. By means of the high-level skeleton tracking method [15], we remove the background and extract the skeleton from the captured images. This tracking method generates the positions of 17 joints, which include head, neck, torso, left and right collars, light and right shoulders, left and right elbows, left and right wrists, left and right hips, left and right knees, and left and right foot. Different from using extra user calibration [4], we infer information about the user's height and body characteristics. It helps us to accurately obtain the changed skeletons in each frame, as shown in Fig. 6.8. We also omit the joints of left and right foot because we consider their weights can be incorporated into shank.

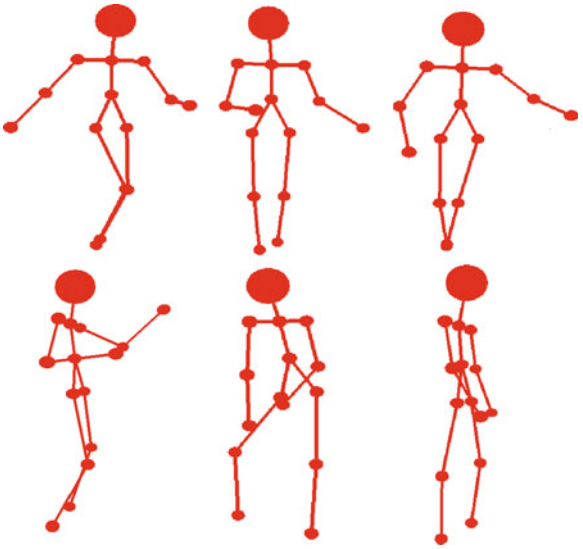

Fig. 6.8 We obtain a depth image of the environment from infrared light sensors, segment, and extract the articulated skeleton from the background. The human body is connected via 15 joints, and the mass is placed on 10 parts (head, torso, left and right forearms, left and right upper arms, left and right thighs, left and right shanks). Compared with original skeleton generated by [15], we omit two joints of feet because we consider their weights can be incorporated into shank. The dance actions captured include walk, hand gesture, jump, rock, and rotation. These actions are represented by a series of equivalent rigid body motions in our method. Therefore, each dance's action can be converted to the movements of body parts, resulting in energy requirements

Table 6.1 We record every part's percentage of the body weight according to the knowledge of human factors engineering. For example, the torso has 58 % body weight

Body part	Weight proportion (%)
Head	23.1
Forearm	1.8
Upper arm	3.5
Thigh	9.4
Shank	4.2
Torso	58

We define a typical human model with 175 cm height and 65 kg weight by a normal relation $W = H - 110$ between height H and weight W. According to human factors engineering, every body part has a percentage of the body weight as listed in the Table 6.1. For example, the torso is 58 weight. In our work, human body contains 10 main mass parts, head, torso, left and right forearms, left and right upper arms, left and right thighs, left and right shanks. They are connected as shown in Fig. 6.8. The weight of neck is incorporated into head, and the weight of left and right foot is incorporated into shank. In the next step, we will use this proportion to compute the power required to move each part.

6.2.3 Energy Consumption Model

Here, we focus on describing the energy consumption model adopted in the application. We use this model to provide dancers with their energy consumption values in each frame and added up each frame's energy consumption to obtain the total consumption. According to the human model previously generated, each dancing action is composed of the motions of all the body parts. Every body part is considered as a rigid part, and these rigid parts are connected by 15 nodes. Energy consumption of each motion is approximately equivalent to the required power to overcome gravitational potential energy of moving parts. The power is obtained by tracking the position change of each body part, and the powers of all the parts are summed up. Certainly, we view each body part as a rigid body with uniform mass distribution.

In order to acquire the kinetic energy of each moving part, its motion is firstly decomposed into two directional movements in vertical plane and horizontal plane. In the model, the vertical coordinate axis is defined as y axis, horizontal coordinate axis is defined as x axis, and z axis points outside the display. We have known that the body is composed of k mass parts $\{m_1, ..., m_i, ..., m_k\}$. Here, k is set to 10 because the human body contains 10 main mass parts mentioned in the previous section. Assume each part has two end joints l, r. The height change of this part between the frame $t+1$ and the frame t is defined as $\Delta h(t)$. The power E_v of moving part m_i in the vertical plane is given by

$$E_v(i) = m_i g \Delta h(t) = m_i g \frac{\Delta y_l(t) - \Delta y_r(t)}{2}. \tag{6.13}$$

where $\Delta y_l(t)$ and $\Delta y_r(t)$ are y coordinate variations of the end node l, r between time $t+1$ and t. Because gravity does not work during fall from the viewpoint of energy consumption, we only take into account the gravitational potential energy while rising. That is, the following condition should be satisfied

$$\Delta h(t) > 0. \tag{6.14}$$

We deduce the required power E_h during movement in the horizontal plane and represent it via the kinetic energy in the horizontal plane

$$E_h(i) = E_x(i) + E_z(i) = \frac{1}{2} m_i (v_x(i)^2 + v_z(i)^2). \tag{6.15}$$

where i means the ith mass part. $E_x(i)$ is the kinetic energy along x-axis, and $E_z(i)$ is the kinetic energy along z-axis. $v_x(i)$ is the moving speed of the mass part in the x direction, and $v_z(i)$ is the speed in the z direction. Here, the speed in x direction is obtained by differentiating its position as follows

$$v_x(i) = \frac{\Delta x_l(t) - \Delta x_r(t)}{\Delta t}. \tag{6.16}$$

where $\Delta x_l(t)$ and $\Delta x_r(t)$ are x coordinate variations of the end nodes l and r between time $t+1$ and t. Similar as the speed calculation in the x direction, the speed in z direction can be obtained by

$$v_z(i) = \frac{\Delta z_l(t) - \Delta z_r(t)}{\Delta t}. \tag{6.17}$$

where $\Delta z_l(t)$ and $\Delta z_r(t)$ are z coordinate variations of the end nodes l and r between time $t+1$ and t.

Finally, the total power $E(i)$ of the ith part between two frames is computed as the summation of powers $E_v(i)$ and $E_h(i)$ in the two planes. We consider that intervals between two frames are uniform, and hence, we see movement of rigid part as uniform linear motion. The total energy consumption E of the body at time t is the energy summation of all the parts.

6.2.4 Experimental Results

We implemented the application based on C++, and the skeleton movements are visualized and confirmed in OpenGL. We programmed the computation process of energy consumption using C++ and visualize the interaction in a gaming development software, Unity [18].

We totally experimented ten dancers' depth images and extracted their skeletons. Every skeleton is used to drive a 3D human, and the joints connect corresponding body parts. The instantaneous and total energy consumption are calculated in real time. We obtained energy consumption in each frame and illustrate the relationship between energy consumption and time. The cumulative energy consumption curves of the first five dancers are drawn in the Fig. 6.9a. Another five dancers' curves are shown in the Fig. 6.9b. Curves with larger slope show that these dancers consume more energy at this frame because their dances need more motions of body parts. In order to interactively visualize the result, we also map the dancer' motions onto a photo-realistic avatar and place the avatar into Unity development environment [18]. And then, the system provides a vivid description of energy consumption, that is, we suggest a conversion of dancers' energy consumption to fat consumption per minute (FCPM). It is known that 1 g fat contains the energy of 37.67 kilojoule. Player's motion interactively gives the dynamic value of fat consumption per minute, for example, 0.427 g, as illustrated in Fig. 6.10.

6.2.5 Conclusion

In this section, we discussed our proposal about a real-time evaluation system of energy consumption while playing an interactive game or dancing. This novel

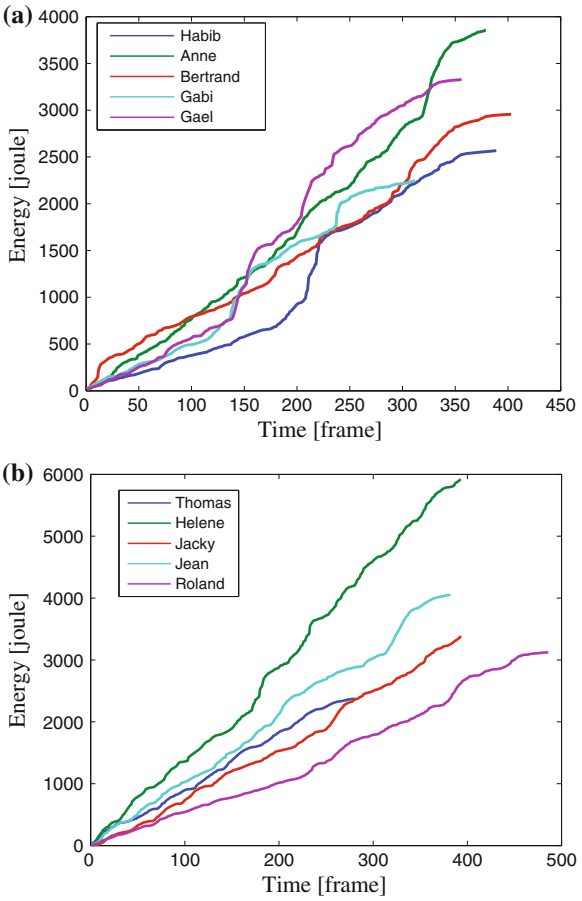

Fig. 6.9 We calculate the cumulative energy consumption curves for all 10 dancers. **a** The cumulative energy consumption curves of first five dancers during the whole dancing process. **b** The cumulative energy consumption curves of another five dancers. The unit of the *horizontal axis* is frame (30 frames per second), and the unit of *vertical axis* is joule. Because the ten dancers' dancing time is different, the horizontal axis of each curve has different length from others. Curves with larger slope show that these dancers consume more energy because their dances need more movements of human body

application could help players to understand their energy status in real time and reduce their weights and plan healthy diet. We tested ten dancers' motions and generated their energy consumption curves stably. The energy consumption is converted into the value of burned fat so that players could interactively know the exercise effects during dance. In the future, we consider introducing a 3D human model construction method by extending the process of 3D freeform model design proposed by Igarashi et al. [22]. The extracted skeleton from depth sensors is dilated into a 3D human model with accurate body proportion, which will be directly placed in the

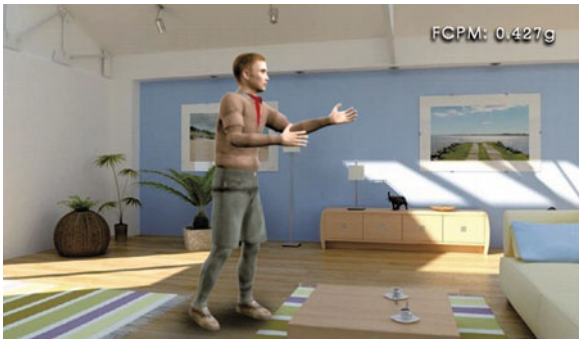

Fig. 6.10 Visualization result in one frame. In each captured image, the power of movements per minute is converted to fat consumption per minute. And fat consumption per minute (FCPM) is displayed in the *left top corner*. The unit of FCPM is gram. Players could interactively know the exercise effect in burning fat in real time

interactive environment to promote user experience. High-quality 3D human model construction with multiple Kinects will be also considered in our future research because the error between real players and a mean human model used in our system is possible to be avoided.

6.3 Efficient Recognition of 3D Human Actions Captured from Kinect

With the development of computer, people want to use computer in a lively and intuitive manner. In order to improve user experience with computers, we require some methods to make computer recognize action automatically. In order to interact with computer by action command, we propose a new real-time system that automatically classifies the human action acquired by consumer-priced RGBD sensor. The main contributions include two effective features extracted from RGBD low-quality videos, average geometric feature and moving body part feature, which represent action in certain degree. Classification experiments show that using these two features, the accuracy of action recognition is acceptable. Users are capable of interacting with computers through human action under Kinect environment.

6.3.1 Introduction

With the development of 3D capturer such as the Microsoft Kinect, users hope to intuitively interact with computers, instead of traditional mouse and keyboard. Compared with the depth of the traditional camera and motion capture system, more

game players and researchers use the Kinect because of its low price. It is possible to use depth sensors like the Kinect to capture human motion in an easy and robust way. This will help many interactive games to animate an avatar and promote user experiences in games.

Existing approaches to human action and gesture recognition can be coarsely grouped into two classes. The first uses 2D image sequences, e.g., Schudlt et al. [27] use bag-of-words representations of the videos that discard much of the spatial structure of the data, which have proved effective in the classification of simple human actions, e.g., walking, running, and boxing. The second approach uses 3D information provides many benefits over the pure image based methods. Using the data of industrial motion capture, Mller et al. [25] transform motions into time-varying geometric feature relationships to achieve low dimensionality and robustness in the matching process. Their approach is scalable and efficient for noise-free motion capture data. Because of high cost of industrial motion capture system, it seems impossible to be widely used. An alternative way is Kinect, which also provides relatively robust motion capture at a low cost. Using the motion capture of Kinect, Raptis et al. [26] propose a complete system which uses a novel angular skeleton representation, a cascaded correlation-based max-likelihood multivariate classifier, and a space–time contract–expand distance metric to process robust action recognition.

A lot of somatic games such as XBOX 360 Kinect sports change the way of game, but these somatic games always focus on motion track rather than motion recognition. Our aim is to expand substantially the interaction with computer, by using simple actions to control the computer. To achieve this objective, we propose a novel system that automatically identifies actions captured by Kinect, which may help users to interact with computer more intuitively and easily. We model 3D human skeleton [15] by connecting different body parts using 15 joints and extract the feature from it. Then, we train a k-nearest neighbors (k-NN) classifier which is used to recognize the action automatically.

Feature is a key to recognize action. **Geometric feature** [24] is used first to convert the spatial information to boolean values. By analyzing geometric feature, we discover two new features: average **geometric feature** and **moving body parts feature**, which have a good representative of action.

The remainder of this section is organized as follows. We describe the proposed features of RGBD data in Sect. 6.3.2. In Sect. 6.3.3, we describe how to use the feature to build a classifier. In Section 6.3.4, we use 83 motion clips in 10 classes to test our method.

6.3.2 Action Feature

The first step of our method is to extract the skeleton from the captured images. We first extract the skeleton from the captured images. Then, its geometric feature will be computed. Based on the geometric feature, we introduce average geometric feature and moving body part feature.

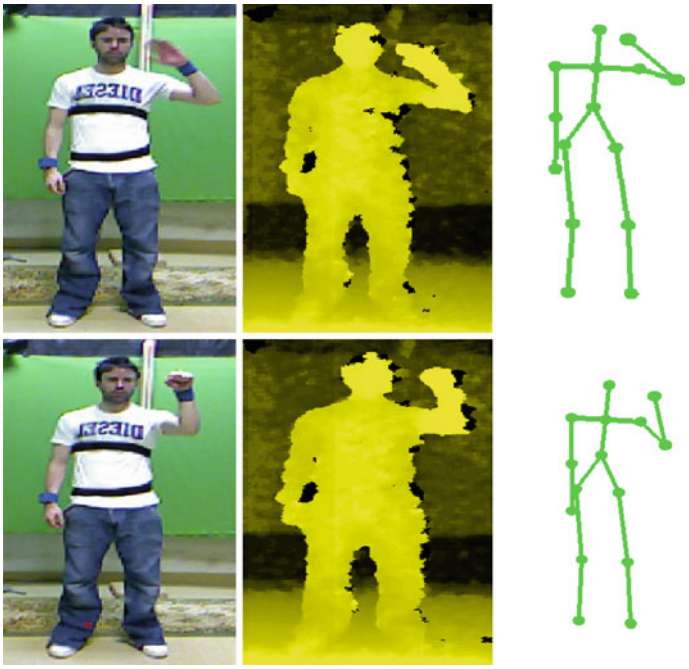

Fig. 6.11 We compute a depth image of the environment from infrared light sensors, segment, and extract the articulated skeleton from the background. The human body is connected via 15 joints. Each image from *left* to *right* represents RGB image, depth image, and skeleton. The *upper row* is a single frame of waving hand, and the *bottom row* is a frame of knocking the door

Kinect Skeleton Tracking. We follow Shotton's [15] method to extract the skeleton from the captured images. Through the depth sensor, we get depth image which is used to process skeleton tracking. We obtain the skeleton with 15 nodes, which include head, neck, torso, light and right shoulders, left and right elbows, left and right wrists, left and right hips, left and right knees, and left and right foot, as shown in Fig. 6.11.

Average Geometric Feature. In order to analyze the motion with fixed standard, we follow the idea of Mller et al. [24] to describe geometric relations between some nodes of a pose. For example, consider a fixed pose, which we test whether the right foot lies in front of (feature value is zero) or behind (feature value is one) the plane spanned by the center of the torso, the left hip joint, and the right hip joint. Another way to calculate geometric feature is measuring the distance of nodes. For example, the distance of both hands is below (feature value is one) or above a certain distance threshold. Threshold is set by hand or position of skeleton nodes. A motion clip with K frames yields a feature matrix G with the size of $K \times D$. $G_i = 0$ or 1, D is the dimension of feature. In our work, D is set to 13. See details in Table 6.2.

The geometric feature used above only contains the information of single frame. For example, the single-frame skeleton in Fig. 6.12, it may be punching, clapping, or

Table 6.2 The geometric feature used in our work

Body part	Threshold	Geometric feature
Arm part	[0° 150°]	Left arms bent
Arm part	[0° 150°]	Right arms bent
Arm part	0.150 m	Left hand approaching left hip
Arm part	0.150 m	Right hand approaching right hip
Arm part	0.200 m	Left hand approaching head
Arm part	0.200 m	Right hand approaching head
Arm part	head height	Left hand raised
Arm part	head height	Right hand raised
Arm part	0.150 m	Hand approaching
Mid part	[15° 180°]	Back bent
Leg part	Right knee height	Left foot raised
Leg part	Left knee height	Right foot raised
Leg part	0.525 m	Foot approaching each other in horizontal plane

We calculate 13-dimensional geometric feature of arm, middle, and leg body parts. The middle column is threshold value. The right column shows the meaning of geometric feature

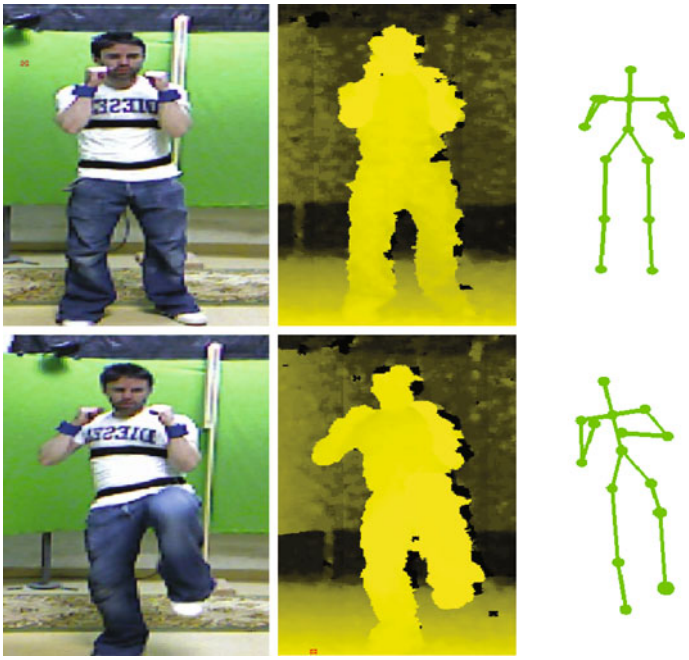

Fig. 6.12 The *upper row* shows single frame of punching action; however, it may be a clapping action or one of other actions. The *lower row* represents single frame of punching and kicking action, and its left leg rises compared to punching action

other actions. It shows that the single frame may be classified incorrectly. Because continuous action can provide much more information, we want to find feature which can distinguish between actions better. The average value of geometric feature can summarize the change rule; therefore, it is a representative feature for action recognition. For example, the waving hand action's arm is always bending, so the average value of geometric feature correspondence to this arm is close to 1; in contrast, the same average feature of the knocking action is about 0.7 because the arm sometimes unbend in this action. We can infer that the average value of geometric feature is a simple but robust feature. After extracting the geometric feature of a motion clip, we calculate the average value AG of every dimension, the element value of AG ranges from zero to one. So a motion clip has an average geometric feature vector which size is D.

Moving Body Part Feature. Each class of action has specific moving parts of body that means each class of action has typical numbers which accounts the number of nonzero geometric features. Actions with different moving body part should not belong to the same class. For example, if legs in one motion clip keep static, while legs in another motion clip have obvious changes, these two motion clips should not belong to the same class, as shown in Fig. 6.12. Our geometric feature is calculated due to the moving degree of body parts, each class may have a special number of moving body parts. Based on this observation, we calculate the number of moving arm parts, central body part, leg parts of every motion clip according to geometric feature, respectively. Each motion clip has the moving body part feature M with the size of 3, as shown in Fig. 6.13. We can conclude that the number of moving body parts in most action classes is constant. A few action classes may have some fluctuation, e.g., in the jumping jacks, and the number of moving arm parts varies. This is caused by the all-body movement. And we can also discover that only in specific action classes, certain body parts have movements. Such as middle body parts, only punch and kick, and weight lift, have changes.

6.3.3 Action Recognition

In this section, we will first train a k-NN classifier by using the features proposed in the above section. Then, we use the trained classifier to realize action recognition. We adopt a k-NN classifier, and the classifier is trained from a set of hand-labeled motion clips. These motion clips contain complete action and incomplete action. The difference between complete action and incomplete action is the length of action. For example, the average number of punching frames is about 50. Hence, a motion clip of punching lasts for 40 frames and can be defined as complete action. However, if a clip has only 20 frames, we classify the clip into incomplete action. Both complete and incomplete action clips will be used for training. Then, the average geometric feature combines with moving body part feature, and accordingly we get a feature vector S with the size of 16. Each feature S of motion clip is regarded as the input

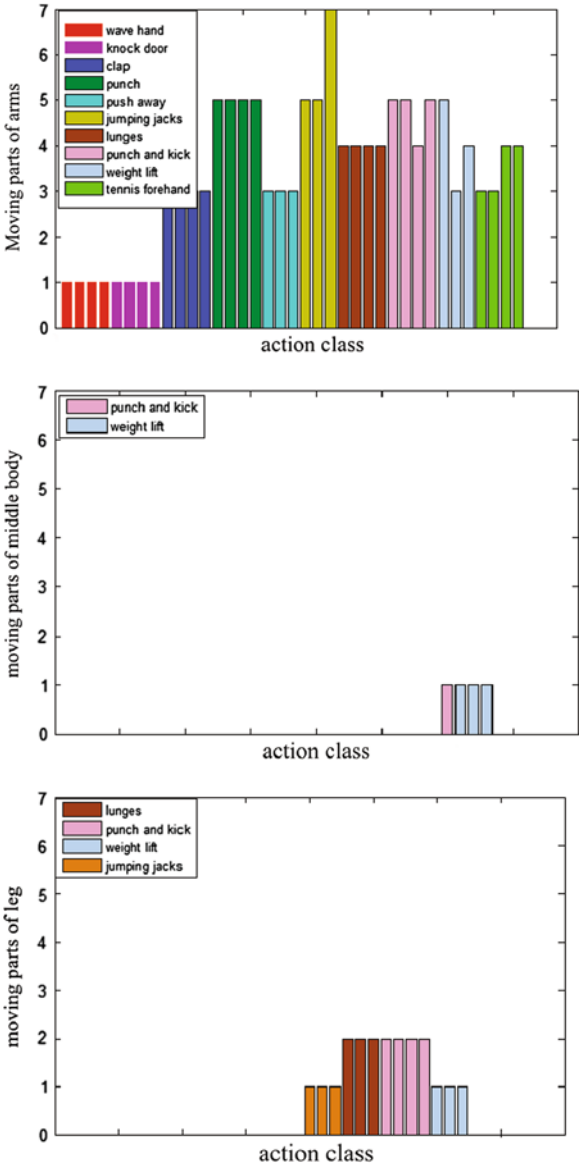

Fig. 6.13 The *upper table* shows the number of moving parts of arms. The *middle table* shows the number of moving parts of middle body. The *bottom* of table illustrates the number of moving parts of legs

of classifier. The classifier parameters are fixed via training and will be employed to recognize novel actions on line.

Once the classifier is trained, labeling motion clip is an automatic process. In order to get a label of motion clip, we extract the features as the same as the training process. Then through examining the labels on each of the closest samples, the label of testing motion clip can be determined. In order to examine our feature further, a SVM classifier is also trained to perform classification. k-NN classifier can be compared with SVM.

6.3.4 Experimental Results

We programmed the algorithm of action recognition using C++. We used the provided dataset captured using a Kinect by 3DLife/Huawei Grand Challenge [10]. Our classifier was trained on 85 motion clips which contain 10 classes. And the testing data are composed of 83 motion clips. Both training data and testing data contain different frames which represent the completeness of the action. The number of nearest neighbors is set to 4 in our work. The average time of extracting feature and recognition by using the motion clip with average 30 frames is shown in Table 6.3. The classification performance is computed in four cases: (1) motion clips with high completeness; (2) motion clips with low completeness; (3) using only the average geometric feature AG; (4) using both the average geometric feature AG and moving body part feature M. The results are summarized in Tables 6.4 and 6.5. We also used SVM classifier to test our two features. The test data contains both incomplete action and complete action. The result is shown in Table 6.6, and the classification accuracy of SVM is clearly lower than that of k-NN.

We can find that (1) by using moving body part feature, the classification accuracy increases by about 10 %. Moving body part feature is validated useful to discriminate the motion which average geometric feature is similar to another. In Fig. 6.12, without moving body part feature, the action of punching and kicking is always recognized as punching, because of the short length of kick action. The average value of leg geometric feature is also small. However, with feature of the body parts, we solved this problem when counting the number of moving parts of the body can discriminate two actions. (2) We compared classification accuracy between motion clips with low completeness and high completeness, the motion clip with high completeness achieved higher accuracy. The incomplete motion clip is similar to single-frame action, which contains ambiguous information resulting in incorrect classification. The result also accords with the rule that the longer action is more possible to be identified, which is similar as the viewpoint of human. (3) Recognition of jumping

Table 6.3 Average time of feature extraction and action recognition

Process	Average time
Feature extraction	0.076 s
Recognition	0.850 s

Table 6.4 The classification accuracy of incomplete motion clip

Action	AG + M (%)	AG (%)
Hand waving	60	40
Knocking door	75	75
Clapping	75	75
Punching	75	75
Push away	25	25
Jumping jacks	100	100
Lunges	100	100
Punching and kicking	66.7	0
Weight lifting	100	100
Tennis forehead	50	75
Average	72.57	64.86

The middle column is the classification accuracy using both average geometric feature AG and moving body part feature M, and the right is the accuracy using only average geometric feature

Table 6.5 The classification accuracy of complete motion clip

Action	AG + M (%)	AG (%)
Hand waving	80	80
Knocking door	25	25
Clapping	75	75
Punching	75	60
Push away	80	40
Jumping jacks	100	100
Lunges	100	100
Punching and kicking	100	50
weight lifting	100	80
Tennis forehead	75	50
Average	80.43	67.35

Table 6.6 The classification accuracy of action

	AG + M (%)	AG (%)
Average accuracy	59.01	38.89

jacks, lunges, and weight lifting obtains very high accuracy. The reason is that both average geometric feature and moving body part feature in these three actions are very different, which leads to higher discrimination. (4) Our method did not perform well in the action of knocking the door illustrated in Fig. 6.11. This action is commonly misclassified as waving hand because their geometric features are close. (5) The time of recognition is fast, and it is possible to implement the real-time applications.

6.3.5 Conclusion

In this section, we proposed a real-time human action recognition system used for Kinect applications. Two new features, average geometric feature and body part feature, were introduced to represent different actions. These features were classified with k-NN classifier to achieve high classification accuracy. However, our method does not perform well in ambiguous action classes. In the future, we plan to extract more robust and discriminative features to improve the accuracy, and structural classifier such as latent structural SVM will also be investigated to enhance classification performance.

References

1. Berger M, Levine JA, Nonato LG, Taubin G, Silva CT (2013) A benchmark for surface reconstruction. ACM Trans Graph 32(2):20
2. Microsoft Kinect (2010) http://www.xbox.com/kinect
3. Han J, Shao L, Xu D, Shotton J (2013) Enhanced computer vision with Microsoft Kinect sensor: a review. IEEE Trans Cybern 43(5)
4. Alexiadis D, Daras P, Kelly P, O'Connor NE, Boubekeur T, Moussa MB (2011) Evaluating a dancer's performance using Kinect-based skeleton tracking. In: Proceedings of ACM multimedia
5. Bleiweiss A, Eshar D, Kutliroff G, Lerner A, Oshrat Y, Yanai Y (2010) Enhanced interactive gaming by blending full-body tracking and gesture animation. In: Proceedings of ACM SIGGRAPH Asia sketches
6. Pedersoli F, Adami N, Benini S, Leonardi R (2012) XKin-: eXtendable hand pose and gesture recognition library for Kinect. In: Proceedings of ACM multimedia
7. Tong J, Zhou J, Liu L, Pan Z, Yan H (2012) Scanning 3D full human bodies using Kinects. IEEE Trans Vis Comput Graph 18(4):643–650
8. Ye G, Liu Y, Deng Y, Hasler N, Ji X, Dai Q, Theobalt C (2013) Free-viewpoint video of human actors using multiple handheld Kinects. IEEE Trans Cybern 43(5)
9. Barmpoutis A (2013) Tensor body: real-time reconstruction of the human body and avatar synthesis from RGB-D. IEEE Trans Cybern 43(5):1347–1356
10. ACM Multimedia 2013 Huawei/3DLife Grand Challenge. http://mmv.eecs.qmul.ac.uk/mmgc2013/
11. Daniel HC, Juho K, Janne H (2012) Joint depth and color camera calibration with distortion correction. IEEE Trans Pattern Anal Mach Intell 34(10):2058–2064
12. Cignoni P, Montani C, Scopigno R (1998) DeWall: a fast divide and conquer Delaunay triangulation algorithm in Ed. Comput Aided Des 30(5):333–341
13. Kazhdan M, Bolitho M, Hoppe H (2006) Poisson surface reconstruction. In: Proceedings of eurographics symposium on geometry processing
14. Lorensen W, Cline H (1987) Marching cubes: a high resolution 3D surface reconstruction algorithm. In: Proceedings of ACM SIGGRAPH
15. Shotton J, Fitzgibbon A, Cook M, Sharp T, Finocchio M, Moore R, Kipman A, Blake A (2011) Real-Time human pose recognition in parts from single depth images. In: Proceedings of IEEE conference on computer vision and pattern recognition
16. OpenCV. http://www.opencv.org
17. http://www.openni.org
18. Unity. http://www.unity3d.com

19. 3DLife/Huawei ACM MM grand challenge. http://perso.telecom-paristech.fr/essid/3dlife-gc-12/
20. Suma EA, Lange B, Rizzo A, Krum DM, Bolas M (2011) FAAST: the flexible action and articulated skeleton toolkit. In: Proceedings of IEEE virtual reality conference
21. OpenGL. http://www.opengl.org
22. Igarashi T, Matsuoka S, Tanaka H (2007) Teddy: a sketching interface for 3D freeform design. In: Proceedings of ACM SIGGRAPH courses
23. 3DLife/Huawei ACM MM Grand Challenge. http://mmv.eecs.qmul.ac.uk/mmgc2013/
24. Mller M, Rder T, Clausen M (2005) Efficient content-based retrieval of motion capture data. ACM Trans Graph 24(3):677–685
25. Mller M, Rder T (2006) Motion templates for automatic classification and retrieval of motion capture data. In: Proceedings of ACM SIGGRAPH/Eurographics symposium on computer animation, pp 137–146
26. Raptis M, Kirovski D, Hoppe H (2011) Real-time classification of dance gestures from skeleton animation. In: Proceedings of ACM SIGGRAPH/Eurographics symposium on computer animation, pp 147–156
27. Schuldt C, Laptev I, Caputo B (2004) Recognizing human actions: a local SVM approach. In: Proceedings of international conference on pattern recognition, pp 32–36