Digitize Your Body and Action in 3-D at Over 10 FPS: Real Time Dense Voxel Reconstruction and Marker-less Motion Tracking via GPU Acceleration

Jian Song*, Yatao Bian*, Junchi Yan, Xu Zhao, Yuncai Liu

Department of Automation, Shanghai Jiao Tong University, and Key Laboratory of System Control and Information Processing, Ministry of Education of China, Shanghai 200240, China

Abstract

In this paper, we present an approach to reconstruct 3-D human motion from multi-cameras and track human skeleton using the reconstructed human 3-D point (voxel) cloud. We use an improved and more robust algorithm, probabilistic shape from silhouette to reconstruct human voxel. In addition, the annealed particle filter is applied for tracking, where the measurement is computed using the re-projection of reconstructed voxel. We use two different ways to accelerate the approach. For the CPU only acceleration, we leverage Intel TBB to speed up the hot spot of the computational overhead and reached an accelerating ratio of 3.5 on a 4-core CPU. Moreover, we implement an intensively paralleled version via GPU acceleration without TBB. Taking account all data transfer and computing time, the GPU version is about 400 times faster than the original CPU implementation, leading the approach to run at a real-time speed.

1 Introduction

Visual information based human motion analysis is an important research direction in computer vision. And 3-D human motion reconstruction and tracking is one of the most challenging tasks. There are huge applications of the related techniques, ranging from virtual realization, film animation, intelligent surveillance, high level human machine interaction, video transmission and compression to sports and medical treatment. With the rapid development of hardware facilities and the theory of computer vision, currently, building a video based human motion construction system for real application is attracting more and more attentions from both academic and industrial fields. Considering the intensive computational demand, GPU will definitely play a very important role in this campaign. This is also the reason why we make our best to take part in this contest.

The main task of 3-D human motion reconstruction is to estimate the motion parameters and body configuration of 3-D human body model, by utilizing visual information from multiple cameras. It also means that we aim to get the position and the configurations of human body parts in real 3-D world. Due to the complexity of this problem, the computational complexities of related algorithms are impermissible to a real-time implementation in CPU. Therefore, the algorithm acceleration using GPU is a nice choice. Nowadays, GPU is getting its wide recognition and application

* denotes equal contribution
among scholars, especially in the 3-D reconstruction field. A latest relevant case is the work presented by Jan-Michael Frahm et al. from UNC, who published their work on ECCV (European Conference on Computer Vision) [3]. They collected the photos from internet (millions of images), thanks to GPU for acceleration, they successfully reconstruct the city of Rome in one day, without the help of cloud computing. In comparison, Sameer Agarwal from University of Washington achieve such performance in their ICCV (International Conference on Computer Vision) paper, owing to the high performance from cloud computing (62 workstations), and the number of images they approach is only 150,000, significantly fewer than the work [3]. From above cases, one can get a clear perspective that GPU is playing very important role in computer vision field.

In our project, there are two classes of core techniques: **3-D voxel reconstruction of human body** and **3-D body pose tracking**. Both of them are time consuming. For 3-D voxel reconstruction, the main algorithm is Shape From Silhouette (SFS). Regarding with SFS techniques, Sofiane Yous et al. leveraged the GPU’s advantage in light tracking, and achieved the real time 3-D reconstruction in some published datasets [5] in 2007. Moreover, Neal Orman et al implemented a free viewpoint rendering system thanks to the acceleration capability brought by GPU [6]. Compared with the traditional SFS methods, here we adopt the method proposed in [1][7] to achieve better robustness, which is insensitive to outliers, while at the price of computational inefficiency. And since we have not found any GPU based implementation of this method, we are exploring the potential high efficiency when GPU is adopted in this project.

After obtaining the 3-D voxel of human body, the next step is to perform human pose estimation and tracking. This is another time-consuming part of this project. In industrial field, 4D View Solution² works on 3-D pose estimation but mainly using parallel CPU. In our project, we select GPU as the accelerator. The core algorithm for pose tracking is Annealed Particle Filter (APF). Regarding with the APF based tracking approach, inherently it is a particle filter algorithm, which has a natural form of parallel computing and therefore it is expected to have a high potential in acceleration.

Towards the aim of 3-D human body voxel reconstruction and 3-D pose tracking, the presented project bears the following aspects:

- Using multiple cameras to obtain visual information of human motion in indoor environments.
- Using human skeleton model as the prior model of human body.
- Achieving generality to different types of human motion.
- Achieving quick and robust solution to 3-D human pose tracking.

In sum, the main algorithms in this project are listed as follows:

- For 3-D human body reconstruction, probabilistic shape from silhouette approach is implemented [1][7].
- For 3-D human pose tracking, annealed particle filter method is used [2].

² A start-up company is born out of Rhone-Alpes National Institute for Research in Computer Science and Control. http://r24085.ovh.net/company.html
We next will introduce these algorithms in detail.

2 Algorithm Design

2.1 System Flowchart

Figure 1: System flowchart

Figure 1 shows the system flowchart. For a given image, we first use probabilistic shape from silhouette (PSFS) [1][7] to get reconstructed voxel of human surface. Then with the information from reconstructed human body, we track human skeleton in an annealed particle filter [2] framework. We will give detailed descriptions of the two algorithm parts in the following sub chapters.

2.2 3-D Voxel Reconstruction

2.2.1 General Reconstruction Methods

Usually the two main methods used for reconstruction are based on silhouette and photometry. Photometry-based methods [9][10][11] are sensitive to camera registrations and have a quite high complexity. Silhouette-based methods are popular for use in multi-camera environments mainly due to their simplicity and computational efficiency.

Shape from silhouette [12][13] is a silhouette-based method that is well studied. It first gets the silhouette using a simple background subtraction method. Figure 2 shows how shape from silhouette works. As perspective projection suggests, the object in the scene should be located inside the three dimensional centrum formed by the camera center and silhouette. It is obvious that an object
can be reconstructed by carving a big region of interest with multi-view images taken from that object.

Figure 2 Generalization of 3-D Reconstruction algorithm [12]

2.2.2 Improved Scheme: Probabilistic Shape from Silhouette

We improved the original shape from silhouette. We use the visual hull of the human body as a basis of our algorithm. While very often silhouettes are used to infer shapes in a two-step process: an individual decision about silhouette occupancy is made on a per-view basis, and then shape is inferred geometrically from all available silhouettes using visual hull methods; yet here we infer the probability of each voxel occupancy by fusing the information from image cues in an incremental updating Bayesian framework as first proposed in the work [7] and further improved by our work [1]. The following algorithm is from our published paper [1].

In order to make full use of the image information from multi-views, we do not segment the binary silhouette of each image on the fly. Instead, we adopt a Bayesian framework to calculate the silhouette probability of each pixel. For a pixel $p$ in image $r$, the posterior probability of representing foreground can be calculated by Bayesian theory:

$$P(F_r^r = 1 | I_r^r) = \frac{P(I_r^r | F_r^r = 1)P(F_r^r = 1)}{\sum_{F_r^r = 0} P(I_r^r | F_r^r)P(F_r^r)}$$

where $F_r^r = 1$ indicates the pixel $p$ in image $r$ represents foreground, otherwise, $F_r^r = 0$. $I_r^r$ denotes the color feature vector of pixel $p$ in image $r$. For simplicity, we model $P(I_r^r | F_r^r = 0) \sim N(I_r^r | \mu_r^r, \sigma_r^r)$, a single Gaussian model. Since the foreground color information is unknown for the first frame, we assume the foreground obeys a uniform distribution:

$$P(I_r^r | F_r^r = 1) \sim U(I_r^r)$$

We choose not to favor any pixel location by setting the prior term $P(F_r^r = 1) = P(F_r^r = 0) = 0.5$. And we obtain the silhouette likelihood map defined as:

$$SLM_r^r = P(F_r^r = 1 | I_r^r)$$
Figure 3 shows silhouette likelihood map of four views from two sequences. There are many holes and noises which cannot be removed using normal shape from silhouette method which results not perfect reconstructed voxels.

Consequently, given a set of images \( \{ I^p \}_n = \{ I_1^p, I_2^p, ..., I_n^p \} \), one can calculate the silhouette likelihood maps \( \{ S^p \}_n = \{ S_1^p, S_2^p, ..., S_n^p \} \).

Similar to the process of obtaining silhouette, given the silhouette likelihood map \( \{ S^p \}_n = \{ S_1^p, S_2^p, ..., S_n^p \} \), we estimate the posterior probability representing occupancy of each voxel in volume of interest as follows:

\[
P(V_i = 1 | \{ S^p \}_n) = \frac{P(\{ S^p \}_n | V_i = 1) P(V_i = 1)}{\sum_{V_i=0} P(\{ S^p \}_n | V_i) P(V_i)}
\]  

where \( V_i = 1 \) denotes the voxel \( i \) is occupied by an object, otherwise, \( V_i = 0 \). The pixel of the silhouette likelihood map \( \{ S^p \}_n = \{ S_1^p, S_2^p, ..., S_n^p \} \) is the projection of voxel \( V_i \). The term \( P(\{ S^p \}_n | V_i = 1) \) in Eq. (3) can be calculated in Eq. (4):

\[
P(\{ S^p \}_n | V_i)
= P(\{ S^p \}_{n-1} | V_i) P(S_n^p | V_i)
= \Pi_{k=1}^{n} P(S_k^p | V_i)
\]

Then the problem of voxel occupancy inference is reduced to model the term \( P(S_k^p | V_i) \) \( (k = 1, 2, ..., n) \) Here we introduce a latent variable \( O_k^p \). \( O_k^p = 1 \) denotes there exits object occluding the voxel \( V_i \) on the viewing line connecting the voxel and its projection \( S_k^p \), otherwise \( O_k^p = 0 \). Thus we get the following equation:

\[
P(S_k^p | V_i) = \sum_{O_k^p} P(S_k^p | O_k^p, V_i) P(O_k^p)
\]

For the term \( P(O_k^p) \), we set \( P(O_k^p = 0) = P(O_k^p = 1) = 0.5 \) indicating with no prior preference; for the term \( P(S_k^p | O_k^p = 0, V_i = 0) \) which indicates that there exist neither voxel \( V_i \) nor other object lying on the viewing line of pixel \( S_k^p \), we have:

\[
P(S_k^p | O_k^p = 0, V_i = 0) = 1 - SLM_k^p
\]

For \( P(S_k^p | O_k^p = 1, V_i = 0), P(S_k^p | O_k^p = 0, V_i = 1), P(S_k^p | O_k^p = 1, V_i = 1) \), which denote there at least exists one voxel lying on the viewing line, we assume:

\[
P(S_k^p | O_k^p = 1, V_i = 0) = SLM_k^p
\]
Finally we filter the probability of voxels occupied by human body and then perform a thresholding process, to remove voxels inside human body and get the surface voxels.

\[
P(S^+_k | O^+_k = 0, V = 1) = SLM^+_k \tag{8}
\]
\[
P(S^-_k | O^-_k = 1, V = 1) = SLM^-_k \tag{9}
\]

2.3 Human Skeleton Tracking

2.3.1 Annealed particle filter

Particle Filter (PF) is based on Monte Carlo methods. It represents the probability distribution by particle set which can be used in any state space model. The main idea of Particle filter is to select random state particle from posteriori. This process is a sequential important sampling. In other words, PF is an Approximation Method for probability distribution function by finding a set of random sample in state space. Although the probability distribution is only an approximate for real distribution, it drops the limitation of random variables satisfying Gaussian distribution in nonlinear filter because of the non-parameter structure. Therefore the particle filter has wider and stronger modeling ability. The procedure of Particle Filter can be seen from Figure 4.
Simulated annealing
Simulated annealing is an approach for minimizing multivariate functions. The term simulated annealing derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure.

The cooling schedule is to anneal the problem from a random solution to a good, frozen, placement. Specifically, we need a starting hot temperature (or a heuristic for determining a starting temperature for the current problem) and rules to determine when the current temperature should be lowered, by how much the temperature should be lowered, and when annealing should be terminated.

Annealed particle filter
The process of Annealed particle filter (APF) [14] is stated as follow:

1. For every time step $T_k$, an annealing run is started at layer $M$, with $m = M$.

2. Each layer of an annealing run is initialized by a set of un-weighted particles $S_{k,m}$.

3. Each of these particles is then assigned a weight

$$\pi_{k,m}^i \propto \omega_s(Z_i, s_{k,m}^i)$$

which are normalized so that $\sum N \pi_{k,m}^i = 1$. The set of weighted particles $S_{k,m}^\pi$ has now been formed.

4. N particles are drawn randomly from $S_{k,m}^\pi$ with replacement and with a probability equal to their weighting $\pi_{k,m}^i$. As the $n^{th}$ particle is chosen it is used to produce the particle $s_{k,m-1}^n$ using

$$s_{k,m-1}^n = s_{k,m}^n + B_m$$

where $B_m$ is a multi-varient Gaussian random variable with variance $P_m$ and mean 0.
5. The set \(s_{k,m-1}\) has now been produced which can be used to initialize layer \(m - 1\). The process is repeated until we arrive at the set \(s_{k,0}\).

6. \(s_{k,0}\) is used to estimate the optimal model configuration \(X_k\) using

\[
X_k = \sum_{i=1}^{N} s_{k,0}^{i} \pi_{k,0}^{i}
\]  

7. The set \(s_{k+1,M}\) is then produced from using \(s_{k,0}\)

\[
s_{k+1,M} = s_{k,0} + B_0
\]

This set is then used to initialize layer \(M\) of the next annealing run at \(t_{k+1}\).

### 2.3.2 Tracking Skeleton using Reconstructed Voxels

10 cylinders are set for different parts of the human body. The following table shows the corresponding part of the cylinders

|   |   |
|---|---|
| 1 | TORSO |
| 2 | Left thigh (upper leg) |
| 3 | Left calf (lower leg) |
| 4 | Right thigh (upper leg) |
| 5 | Right calf (lower leg) |
| 6 | Left upper arm |
| 7 | Left lower arm |
| 8 | Right upper arm |
| 9 | Right lower arm |
| 10 | Head |

Table 1: 10 cylinders of human
We can get articular human model by setting base and bottom radius and length for each cylinder. This model has 31 degrees of Freedom, representing position and freedom of the ten cylinders.

Considering the 31 DOF of skeleton in one frame as a particle vector, and using the projection of voxels to be the prior knowledge, we can find an optimized particle and get the new skeleton in the annealed particle filter framework. First we propagate the particles in the 31 dimension space. Each of the particles represents one pose of a skeleton. Then we match images generated by these particles and the projection of voxels in 2-D images and also their edges. We use the matched value as weight numbers for particles and we get a new particle after adding all these weighted particles. This new particle is then pushed into the next annealing layer. After propagating particles in all layers, we get the optimized particle, which represents the human skeleton in a new frame. Particle number is usually set to a value from tens to hundreds depending on the tracking condition. And the annealing layer is usually set to around 10.

**Particle Weight**

Weight value in formula (10) is divided into two parts. One is from the edge information,

\[
\sum e_i(X, Z) = \frac{1}{N} \sum_{i=1}^{N} (1 - p^e_i(X, Z))
\]

And the other is from the silhouette,

\[
\sum r_i(X, Z) = \frac{1}{N} \sum_{i=1}^{N} (1 - p^r_i(X, Z))
\]

While we have multi-cameras in reconstruction and tracking, the final weight is a sum of that two parts from all cameras, that is

\[
\omega(X, Z) = \exp\left(-\sum_{i=1}^{C}\left(\sum r_i(X, Z) + \sum r_i(X, Z)\right)\right)
\]

Where C indicates the camera number.

### 3 Algorithm Parallelization

#### 3.1 Parallelism Analysis

Our scheme is based on computing and processing of images (digitized as matrix in practice), which has high parallelism to the task itself.

From the micro point of view, many intermediate processes are math operations of images or matrices. In the algorithm, large scale data is stored as continuous large array (2-dimensions or 3-dimensions). These math operations can be accelerated by GPU parallelization.

From the macro point of view, both reconstruction and tracking parts in our algorithm are the procedures of processing multi-camera information. These separate processes of each camera can be accelerated by parallelization.
The specific analysis is presented in the following sub chapters according to the nature character of reconstruction and tracking algorithm.

3.1.1 3-D Voxel Reconstruction
The hot spot of 3-D reconstruction is the computation of voxel. And the color rendering is also an iterative process of all voxels.

Some parallelizable parts of the algorithm are listed as follows:

- Reconstruction problem is the procedure of analyzing 3-D space information after image processing, the data of which is multi-dimension matrix whose math operations has parallelism.
- Both reconstruction and color rendering are superposition of processing procedures of multi-cameras which are independent to each other and lead to parallelism.
- Reconstruction is the procedure of filtering all the voxels in three dimension region, the procedures for each voxel has nothing to do with others while the number of voxel is very large. Camera image information is processed in this procedure meanwhile. Great acceleration can be achieved via parallelization of voxel and image pixels processing.
- Color rendering aims to get color information by processing the reconstructed voxels. The large number of voxel and independence of each procedure for one voxel will largely contribute to acceleration via parallelization.

3.1.2 Human Skeleton Tracking
The hot spot of human skeleton tracking lies in the weight assigning process.

Some parallelizable parts of the algorithm are listed as follows:

- Tracking problem is the procedure of analyzing matching degree between real world images and constructed images after image processing, the data of which is multi-dimension matrix whose math operations has parallelism.
- The particles’ weights are computed by formula(10), which will have a significant acceleration due to the large number of particles and the independence of each weight computing procedure.
- In the procedure of assigning weight for one specific particle, formula(16) is a superposition process from several cameras. And the independence of each camera’s processing leads to parallelism.

3.2 Parallelization on Multi-Core CPU
To parallelize our program via GPU, we first parallelized it using multi-threads on multi-core CPU.

The parallelism on CPU is different with that on GPU since CPUs cannot hold as many threads as GPU running at the same time and the thread control is done explicitly. So we only paralleled the top level of all parallelizable parts.

We use Intel® Threading Building Blocks (Licensed under GPL.v2 with the runtime exception) to build parallel programs on CPU.

In the voxel reconstruction part, we parallelized the calculating process of each camera while constructing voxels and rendering colors. And in the human skeleton tracking part, we parallelized the time-consuming process of each particle. These paralleled program parts are represented under the name of XXX_Invoker.
3.3 Parallelization on GPU

3.3.1 Algorithm Restructuring
GPU and CPU are adept in different computational demand scenarios because of architecture differences. Taking into account these differences and in accordance with the GPU computing features, we refined the algorithm structure. The overall frameworks of CPU and GPU algorithms are shown in the following figure.
Figure 6 Comparison of CPU and GPU frameworks for 3-D voxel reconstruction algorithm
Figure 7 Comparison of CPU and GPU frameworks for human skeleton tracking algorithm
The comparisons of CPU and GPU algorithms are shown in Figure 6 and Figure 7, in which yellow background highlights GPU computing parts. For the parts which are more suitable for CPU processing, we modify interfaces and retain CPU implementation to achieve the overall optimum.

As shown in the figures, the algorithm has three major structural changes, that is, one merger and two splits.

- **Merging Spatial Points Smoothing and Voxel Generating**
  In the spatial smoothing process, a float array with maximum length of 2250000 that is 8.6 MB of data is traversed and a new array with the same size is generated which is traversed later to generate filtered voxel cloud. Time complexity of each array traversing is same and memory copy is cheap when doing this on CPU. But for the sake of global memory access limitation on GPU, additional write and read operation of all the data in array will take much time when traversing the array twice other than once, so merging this procedure will significantly improve GPU computing efficiency.

- **Splitting Voxel Color Computation**
  Voxel color calculation can be divided into two parts. Similar to the above analysis, the first half of which could share the same traverse process with voxel projection to reduce global memory access, and because of the different array lengths traversed in the first part and the second part, splitting will not increase memory access. So we split voxel computation into two parts handled by two kernels, respectively.

- **Splitting Skeleton Projection and 2D Cylinders Drawing**
  In this step, the computed 3-D skeleton is projected to corresponding 2D planes of each camera and drawn to the images respectively. In CPU code, 2D cylinders on each image are drawn right after the end of computing skeleton projection in corresponding plane, while in GPU code, 2D cylinders on these images are not drawn until projection of all camera planes are completes. This is because projection computation can be parallelized on GPU and drawing on CPU does not spend lots of time. Moreover, drawing on CPU will refrain from two steps of large data copy.

### 3.3.2 Data Transfer Optimization

Data transfer always takes a long time in GPU program. We use the unblock way to read or write data to GPU device, and use multiple commandqueues to deal with different operations.

Read/write operation is arranged to the same commandqueue when host program needs to read/write data from device and the data will be used by the following kernel or CPU code. And if the data read/write from device is not needed immediately, the operation is put into another commandqueue, and the operations need these data wait for the events from read/write operation.

This will parallelize the CPU computation, GPU computation and data transfer as possible, to hide the time spent on CPU by serial code and the time spent on transferring data between host and device. The timeline after optimization is shown in Figure 8.
3.3.3 Main Program Optimization

Voxel Computation

As shown in Figure 6 this computation has 5 steps. Let \( xlen \) (maximum value is 150) denotes sample numbers of x coordinate, \( ylen \) (maximum value is 150) denotes sample numbers of y coordinate, \( zlen \) (maximum value is 100) denotes sample numbers of z coordinate, and \( camNum \) denotes camera number in use (it is set to be 8 in the program).

a) Compute spatial point coordinates. Compute spatial point coordinates using point location. Global work-item number is \( xlen \times ylen \times zlen \), work-group size is set as \((4, 4, 4)\).

b) Compute silhouette probability map. It is computed separately for each camera, global work item number is image’s pixel number.

c) Compute spatial points’ probability. Compute points’ probabilities using foreground probabilistic graph. The computation result for each camera is reduced using local memory. Global work-item number is \( xlen \times ylen \times zlen \times camNum \). Work-group size is \( (256 / camNum, camNum) \) and work-group number is \( (\text{ceil}(xlen \times ylen \times zlen / (256 / camNum)), 1) \).

d) Smooth spatial points and generate voxel. Global work-item number is \( xlen \times ylen \times zlen \), and work-group size is \((4, 4, 4)\).

e) Remove inner voxel. About 30000 numbers of surface voxel of human is sifted from a large amount of data with size of \( xlen \times ylen \times zlen \times 8 \), which can be up to 18000000. This process is time-consuming on GPU so CPU implementation is reserved.

Voxel Projection and Color Computation

To make full use of GPU resources, we changed the code structure of color computation. Because of the optimization of algorithm implementation, we reduced time complexity without disturbing the correctness of the procedure. So this part of algorithm has a great speed up after the whole optimization on GPU. As shown in Figure 6, procedures in CPU program are split and disordered
into 3 steps after optimization. The global work-item number is xlen * ylen * zlen * camNum in the first step and equals to image pixel number in the following two steps.

**Compute Sum-squared Difference (SSD)**

This part takes most of the time in skeleton tracking algorithm and has the most parallelism as well. As shown in Figure 7, it has 4 specific steps. Let cylNum denotes cylinder number (10 in the program), camNum denotes camera number (8 in the program), particeNum denotes particle number (can be adjusted according to different data, 200 for test data).

a) Compute Spatial 3-D Cylinders. Ten cylinders’ spatial locations and orientations are computed according to the 31 parameters of each particle. The global work-item number is set as particleNum.

b) 3-D Cylinders Projection. Project spatial 3-D cylinders to the corresponding 2D plane of each camera. Global work-item number is particleNum * camNum * cylNum. One specific work-group is in charge of computation of one particle, work-group size is (camNum, cylNum), and work-group number is (particleNum, 1).

c) Compute Edge SSD. Edge SSD is computed using cylinder projections on 2D planes and edge maps. Global work-item number is set as particleNum * camNum * cylNum * 2. The computation of one specific particle lies in one work-group, and then data reducing is issued using local memory. The work-group size is (camNum, cylNum, 2), and work-group number is (particleNum, 1, 1).

d) Compute Silhouette SSD. Silhouette SSD value is computed using cylinder projections on 2D planes and silhouette maps. Global work-item number is particleNum * camNum * cylNum. One specific work-group is in charge of computation of one particle, local memory is used to reduce data. The work-group size is (camNum, cylNum), and work-group number is (particleNum, 1).

3.4 Parallelization Result and Performance Analysis

- **IDE**
  Microsoft Visual Studio 2008, 32 bit MSVC compiler

- **3rd Party Library**
  OpenCV 2.3RC
  QT Library 4.7.3
  Intel TBB3.0 update 7

- **Test Environment**
  CPU: Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz
  RAM: 4.00G DDR3
  GPU: AMD Radeon HD 6870
  Driver date 2011/7/7, version 8.872.0.0
  OS: Windows 7 x64 Enterprise SP1
  OpenCL 1.1 AMD-APP-SDK-v2.5 (684.213)
CAL 1.4.1457 (VM)

- Test Data and Algorithm Parameters
  Data: PEAR Dataset, Subject1_Walk1_Validate, 200 frames

  Particle Number: 200

  Annealed Layer Number: 10

We have implemented the CPU version, CPU + TBB parallel version and GPU version of algorithms, and following paragraphs compare these algorithms to evaluate the performance of GPU acceleration.

Because of the algorithm overlapping in optimization, we are not able to evaluate our algorithms individually. The following analysis takes the whole algorithm procedures into account, including voxel reconstruction, voxel color computation and human skeleton tracking.

TBB is a tool similar to OpenMP, the algorithm paralleled with TBB is essentially the same with the one not using TBB. The only difference is that TBB will automatically create numbers of threads when calculating for loops according to different CPU core number in runtime. Acceleration is achieved by using all available CPU cores. TBB is not used in the GPU accelerated algorithm, as all parallelism is performed on GPU.

In the code submitted in preliminary contest, the default algorithm running on CPU is the CPU+TBB version, which is paralleled using TBB. And due to the sake of automatically assigning threads according to the numbers of CPU cores in TBB, the runtime speed can change exponentially in this version of program when using different CPUs, especially those with different numbers of cores. This time change is not linear because not all parts of the algorithm are calculated using TBB. We use a 4-core Intel CPU to test the TBB-paralleled algorithm, and the result is used to illustrate relative acceleration of GPU program.

|          | Time 1 (ms) | Time 2 (ms) | Time 3 (ms) | Average (ms) |
|----------|-------------|-------------|-------------|--------------|
| CPU      | 33948.37    | 33924.86    | 33759.48    | 33877.57     |
| CPU+TBB (4 cores) | 9605.077    | 9597.044    | 9593.483    | 9598.535     |
| GPU(pre) | 219.8729    | 217.0209    | 218.3901    | 218.428      |
| GPU(final)| 84.53274    | 85.58835    | 84.41453    | 84.845207    |

Table 2 average time per frame (results of 3 times of running)

Table 2 shows the average time consumed per frame of the algorithm running a total 200 frames. The table lists the result of running the algorithm for 3 times as well as the average value. Time consumed on GPU has two versions which represent the time of previously submitted program and the final program. More detailed values of the time cost by each frame can be found in Table 4 in the appendix.

The algorithm takes an average of more than 30 seconds per frame on CPU, while the GPU speed up to about 200 milliseconds per frame, which is close to real time. For scenes with non-violent movements (such as walking), the speed of 200 milliseconds per frame or 5 fps is rather a real-time performance.
The specific acceleration ratios can be computed from Table 2, which are listed in Table 3.

|                      | Acceleration Ratio (pre) | Acceleration Ratio (final) |
|----------------------|--------------------------|----------------------------|
| CPU+TBB (4 cores) / CPU | 3.5295                   | 3.5295                     |
| GPU / CPU+TBB (4 cores) | 43.9437                  | 113.1300                   |
| GPU / CPU             | 155.0972                 | 399.2868                   |

Table 3 Acceleration ration of the algorithm

CPU program parallelized using TBB on 4-core CPU has an acceleration ratio of 3.5, which varies with CPU core numbers.

In the previously submitted program, the acceleration ratio of GPU program compared to CPU program parallelized using TBB on 4-core CPU is a speed up of 44 times, varies with CPU core numbers as well. And the GPU program speeds up about 155 times compared to the original CPU program without TBB parallelism.

The final version of algorithm has an acceleration ratio of GPU program compared to CPU program parallelized using TBB on 4-core CPU of 44 times, varies with CPU core numbers as well.

The final GPU program speeds up about 399 times compared to the original CPU program without TBB parallelism.

4 Prospect

Kinect, a latest product brought out by Microsoft (refer [8] for algorithm details) is one of the paragons for the commercialization of such technique. On the other hand, the startup companies like the “4D View Solution” as mentioned in the previous context are trying to provide the end to end solutions to customers, which to some extent prompt its application in animation, communication, sports and surveillance and such. Based on the existing solutions and methods, one can observe that this technology exhibits a bright prospect based on a solid theory, while the high performance computing platform is still under its infancy. We have tested our reconstructing and tracking algorithm on PEAR database, which originally consumed a lot of time and now has been accelerated to real-time via GPU. Our algorithm running on multi-camera systems or Kinect systems can play a role in animation, communication, surveillance and such areas.

References

[1] Junchi Yan, Yin Li, Enliang Zheng, Yuncai Liu: An accelerated human motion tracking system based on voxel reconstruction under complex environments. Asian Conference on Computer Vision. (2009)

[2] Alexandru O. Balan, Leonid Sigal, Michael J. Black: A quantitative evaluation of video-based 3D person tracking. ICCCN '05 Proceedings of the 14th International Conference on Computer Communications and Networks.

[3] Jan-Michael Frahm, Pierre Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp Svetlana Lazebnik, Marc Pollefeys: Building Rome on a Cloudless Day. European Conference on Computer Vision. (2010)

[4] Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building Rome in a day. In: ICCV.
 Sofiane Yous, Hamid Laga, Masatsugu Kidode, Kunihiro Chihara: GPU-Based Shape from Silhouettes. Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia. (2007)

 Neal Orman, Hansung Kim, Ryuuki Sakamoto, Tomoji Toriyama, Kiyoshi Kogure, and Robert Lindeman: GPU-Based Optimization of a Free-Viewpoint Video System. Electronic Letters on Computer Vision and Image Analysis 7(2):120-133, 2008.

 Franco, J., Boyer, E.: Fusion of multi-View silhouette cues using a space occupancy grid. In: Proc. of ICCV 2005, Beijing, China, vol. 1, pp. 1747–1754 (2005)

 Neal Orman, Hansung Kim, Ryuuki Sakamoto, Tomoji Toriyama, Kiyoshi Kogure, and Robert Lindeman: GPU-Based Optimization of a Free-Viewpoint Video System. Electronic Letters on Computer Vision and Image Analysis 7(2):120-133, 2008.

 Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman and Andrew Blake: Real-Time Human Pose Recognition in Parts from Single Depth Images. Conference on Computer Vision and Pattern Recognition. (2011) CVPR’11 best paper award.

 K. Kutulakos and S. Seitz. A theory of shape by space carving. IJCV, 38(3):199–218, 2000.

 B. Goldluecke, M. Magnor. Space-time isosurface evolution for temporally coherent 3D reconstruction. CVPR2004.

 G. Vogiatzis, PHS. Torr, R. Cipolla. Multi-view stereo via volumetric graph-cuts. CVPR2005.

 G. Cheung, T. Kanade, JY. Bouguet, M. Holler. A real time system for robust 3D voxel reconstruction of human motions. CVPR2000.

 Laurentini, A.: The visual hull concept for silhouette-based image understanding. PAMI 16 (1994)

 J. Duetscher, A. Blake, I. Reid, "Articulated Body Motion Capture by Annealed Particle Filtering," cvpr, vol. 2, pp.2126, 2000 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’00) - Volume 2, 2000

 Appendix

 Test results of running each algorithm 3 times.

| frame | CPU (ms) | CPU + TBB (4 cores) (ms) | GPU (ms) |
|-------|----------|--------------------------|----------|
|       | 1  2  3  | 1  2  3                 | 1  2  3  |
| 1     | 35600.5 | 35652.8                | 35513.9 | 145.425 | 152.239 | 155.904 |
| 2     | 33641.8 | 33700.1               | 33526 | 9470.22 | 9486.45 | 9465.98 |
| 3     | 32839.9 | 32920.3               | 32742.7 | 9219.91 | 9252.42 | 9213.83 |
| 4     | 32906.7 | 32995.6               | 32810 | 9126.02 | 9122.35 | 9103.10 |
| 5     | 33184.8 | 33375.6               | 33204.6 | 9325.89 | 9269.03 | 9330.55 |
| 6     | 33937.3 | 33993.8               | 33826 | 9443.41 | 9358.85 | 9341.25 |
| 7     | 34964.3 | 34996.6               | 34828.6 | 9556.24 | 9535.68 | 9492.39 |
| 8     | 34794.8 | 34836.6               | 34660.4 | 9503.72 | 9474.04 | 9622.65 |
| 9     | 34388.3 | 34414.5               | 34260.6 | 9409.42 | 9324.38 | 9356.72 |
| 10    | 36161.4 | 36156.4               | 36014.8 | 9802.5 | 9958.93 | 9822.64 |
| 11    | 36471.5 | 36427.7               | 36290.7 | 9972.38 | 9936.55 | 9838.54 |
| 12    | 34901.2 | 34903.5               | 34751.5 | 9450.02 | 9569.22 | 9540.04 |
| 13    | 34684.8 | 34648.2               | 34480.4 | 9455.61 | 9445.71 | 9437.74 |
| 14    | 34504   | 34501.8               | 34355.1 | 9462.61 | 9421.62 | 9450.51 |
| 15    | 33844.9 | 33832.5               | 33656.4 | 9305.92 | 9358.47 | 9362.19 |
| 16    | 32626.2 | 32625.8               | 32473.8 | 9057.59 | 8940.46 | 9098.72 |
| 17    | 32540.9 | 32532.1               | 32377.7 | 9027.06 | 9039.79 | 9022.08 | 62.7196 | 78.6232 | 106.246 |
| 177 | 35859 | 35827.6 | 35634.4 | 9978.46 | 9839.39 | 9910.85 | 120.921 | 81.963 | 71.0025 |
| 178 | 34434.8 | 34416.4 | 34213.4 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 179 | 34040.8 | 33897.8 | 33707.6 | 9394.67 | 9391.61 | 9398.92 | 88.7891 | 111.574 |
| 180 | 33710.7 | 33710.2 | 33514.2 | 9547.77 | 9442.56 | 9346.1 | 89.9033 | 76.7708 |
| 181 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 182 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 183 | 34352.9 | 34271.5 | 34061.7 | 9746.24 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 184 | 34484.9 | 34452.1 | 34271.1 | 9844.01 | 9772.22 | 9718.13 | 122.258 |
| 185 | 33710.7 | 33710.2 | 33514.2 | 9547.77 | 9442.56 | 9346.1 | 89.9033 | 76.7708 |
| 186 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 187 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 188 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 189 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 190 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 191 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 192 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 193 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 194 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 195 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 196 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 197 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 198 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |
| 199 | 33442.2 | 33404.4 | 33220.5 | 9584.53 | 9633.77 | 9517.88 | 102.878 | 82.7865 |
| 200 | 33249.5 | 33113.5 | 32925.1 | 9574.77 | 9633.77 | 9517.88 | 88.7891 | 111.574 |

Table 4 time of each frame (results of 3 times of running)