A multi-GPU accelerated virtual-reality interaction simulation framework

Xuqiang Shao1*, Weifeng Xu1, Lina Lin2, Fengquan Zhang3

1 School of Control and Computer Engineering, North China Electric Power University, Baoding, Hebei, China, 2 92524 troops, Ningbo, Zhejiang, China, 3 School of Computer Science, North China University of Technology, Beijing, China

* shaoxuqiang@ncepu.edu.cn

Abstract

In this paper, we put forward a real-time multiple GPUs (multi-GPU) accelerated virtual-reality interaction simulation framework where the reconstructed objects from camera images interact with virtual deformable objects. Firstly, based on an extended voxel-based visual hull (VbVH) algorithm, we design an image-based 3D reconstruction platform for real objects. Then, an improved hybrid deformation model, which couples the geometry constrained fast lattice shape matching method (FLSM) and total Lagrangian explicit dynamics (TLED) algorithm, is proposed to achieve efficient and stable simulation of the virtual objects’ elastic deformations. Finally, one-way virtual-reality interactions including soft tissues’ virtual cutting with bleeding effects are successfully simulated. Moreover, with the purpose of significantly improving the computational efficiency of each time step, we propose an entire multi-GPU implementation method of the framework using compute unified device architecture (CUDA). The experiment results demonstrate that our multi-GPU accelerated virtual-reality interaction framework achieves real-time performance under the moderate calculation scale, which is a new effective 3D interaction technique for virtual reality applications.

Introduction

Natural and real-time 3D interaction with computer-generated virtual environment is an important research topic of virtual reality and augmented reality technologies, as it increases the participant’s sense of immersion. Up to now, there have been many practical application examples of 3D interaction framework based on the haptic feedback devices, especially in the more and more commonly used virtual surgery area. For example, Bro-Nielsen et al. [1] developed a virtual surgical simulator for training the removal of a shattered kidney by open surgery, which allows the users to probe and cut the virtual soft tissues by using the SenSable Technologies Phantom haptic feedback device. By supporting haptic feedbacks through Phantom Omni devices, Webster et al. [2] proposed a realistic, easy to operate, and economical prototype of haptic suturing simulator to train basic suturing operations for simple wound closure. To support complex virtual cutting interactions, Bielser et al. [3] designed a realistic
open surgery trainer which could basically model various interactions of the elastic soft tissue and the virtual scalpel or hook controlled by the force feedback device. Furthermore, to achieve more realistic virtual cutting interactions with bleeding effects, Rianto et al. [4] incorporated GPU accelerated fluid dynamic simulation into a heart surgery simulator exploiting haptic feedback devices. Thus when the surgical knife cut the heart muscle surface in a certain thickness, the blood flowing on the surface of a beating heart is realistically modeled in real time.

Although the above 3D interactions with virtual environments are realistic and immersive, haptic feedback devices are too cumbersome and expensive for common users. With the rapid development of image-based 3D reconstruction technology, camera-based virtual-reality interaction has become a new type of effective 3D interaction technique for virtual reality applications. Up to now, a variety of image-based 3D reconstruction algorithms such as structure from stereo [5], structure from motion [6], shape from shading [7], and shape from silhouettes [8], have been developed. Due to computing the intersection volume of all viewing cones, the 3D reconstruction method from silhouettes is also regarded as visual hull method. Visual hulls are divided into two categories: surface-based visual hull (SbVH) [8] and voxel-based visual hull (VbVH) [9]. SbVH method, which generates 3D polyhedral intersections, is relatively inefficient for the special-case problems, and suffers from numerical instabilities. VbVH method, by contrast, is not susceptible to numerical difficulties that may arise in the SbVH method, and suitable for being accelerated by CUDA. Based on VbVH method, INRIA developed a platform called Grimage [10] which provides the realistic simulation of the interactions of reconstructed objects and computer-generated virtual objects. When any real objects are put into the interaction space, their reconstructed 3D models are instantaneously computed and placed into a virtual environment to interact with computer-generated rigid or elastic objects. However, VbVH whose reconstruction accuracy depends on the number of cameras is only the approximation of the real object. For each point of the image, a time-of-flight (TOF) camera measures the time of flight of a light signal between the camera and the subject. Through refining the reconstructed VbVH by the depth map from a TOF camera, Wang et al. [11] presented a real-time reconstruction platform implemented on the GPU. This platform supports the reconstructed person model pushing, catching and squeezing the virtual rigid objects. Although VbVH-based virtual-reality interaction technology has achieved natural and real-time 3D interaction experiences for the users, in our opinion, there are still several problems to be solved:

1. The 3D reconstruction precision of VbVH method is not high enough due to the occlusion between the models, which leads to unnatural virtual-reality interactions for the users.
2. The supported types of virtual-reality interactions are limited, because the virtual objects are rigid and do not support topology changes.
3. For complex virtual-reality interaction scenarios, the time-consuming computation loads of each time step limit its real-time or interactive applications.

In order to solve the above problems to improve the interaction experiences, based on our previous works on hybrid deformation model for virtual cutting [12], fluid-solid coupling [13] and visual hull reconstruction [14], we develop a type of image-based virtual-reality interaction framework implemented on multiple GPUs using CUDA, which supports more types of interactions including virtual cutting and achieves higher computational efficiency compared to other similar methods. Specifically, compared to other camera-based virtual-reality interaction framework, the main contributions of this paper are summarized as follows:
1. Based on an extended VbVH algorithm, we develop a more precise image-based 3D reconstruction platform for real objects.

2. An improved hybrid deformation model, which couples the geometry constrained fast lattice shape matching method (FLSM) and total Lagrangian explicit dynamics (TLED) algorithm, is proposed to achieve efficient and stable simulation of virtual objects’ elastic deformations.

3. One-way virtual-reality interactions including soft tissues’ virtual cutting with bleeding effects are successfully simulated.

4. With the purpose of significantly improving the computational efficiency of each time step, we put forward an entire multi-GPU implementation method of the framework using CUDA.

The experiment results demonstrate that our virtual-reality interaction simulation framework achieves 25 FPS under the moderate calculation scale, which provides a new effective 3D interaction technique for real-time virtual reality applications.

Materials and methods

System architecture

Based on the previous research work on image-based 3D reconstruction [11, 12, 14, 15], we design a new multi-GPU accelerated virtual-reality interaction simulation framework. The system architecture of our virtual-reality interaction framework is shown as Fig 1: In order to obtain the images of objects to be reconstructed from different perspectives, as shown in Fig 2, 8 high-speed optical cameras and 1 TOF depth camera are placed on a 1m×1m×1m or 3m×3m×3m sized frame in different positions. Each camera is connected to a single PC. Before the system runs, the geometric calibration of all these cameras are implemented by using Zhang’s method [16] to get the intrinsic and extrinsic camera parameters, and the depth calibration of TOF camera is implemented by using Wang’s method [11] which constructs lookup table to correct system error. The synchronization of multiple heterogeneous cameras is realized by using the external triggering mechanism of [11] which sends both high and low level signals.
Signals to the camera signal interface alternately. As shown in Fig 1, the clients, the server, signal trigger, and the TOF camera are connected together by a switch. The Client/Server mode is adopted to construct the system framework. When acquiring the images from their connected cameras, these PCs locally execute the background subtraction method [17] to extract the silhouette of the objects to be reconstructed. Once the silhouettes are computed, each one is sent to the server which is a high performance graphic workstation with multiple GPUs. The multi-GPU server is in charge of parallel 3D reconstruction of real objects, virtual objects' deformation simulation and virtual-reality interaction.

**Ethics statement**

The individual in this manuscript has given written informed consent (as outlined in PLOS consent form) to publish these case details.

**3D reconstruction of real objects**

To reconstruct more accurate objects from the captured images, we put forward an extended VbVH method [15] for 3D reconstruction which exploits the data of TOF depth camera to correct the triangulated surface generated by marching cubes (MC) algorithm. There are three modules for our extended VbVH 3D reconstruction method: (1) visual hull computation, means intersection testing of viewing cones to determine the status of each voxel (Fig 3(a)); (2) triangulation corrected by depth data, means combining depth map data and MC algorithm [18] to extract the triangular mesh surface of the object from the visual hull (Fig 3(b)); (3) texture mapping, means determining the texture source of each triangle for realistic surface rendering (Fig 3(c)).

In our method, the principle of visual hull computation is: (1) in the preprocessing stage, the acquisition space is divided into a series of independent voxels with a certain size. For all voxels, we create a mapping which projects their three-tuple indices as the three dimensional coordinates in the reference framework of TOF camera; (2) in the runtime stage, the occupancy status of each voxel is computed to estimate the reconstructed shape of the real object.
Each voxel has one and only one status which belongs to the status set \{Inside, Outside, Uncertain\}. We determine the status of each voxel according to the following rules.

If the projection of a voxel on each imaging plane lies completely inside the corresponding silhouette, the status of the voxel is defined as Inside;

If the projection of a voxel on each imaging plane lies completely outside the corresponding silhouette, the status of the voxel is defined as Outside;

Otherwise, it means that the status of the voxel can not be defined under the current computational precision, so we define the status of the voxel as Uncertain.

For the object to be reconstructed, all the voxels whose statuses marked as Inside constitute its inside, and all the voxels whose statuses marked as Uncertain contain its boundary. However, the voxel-based volumetric representation of the visual hull can not exactly reflect the boundary information of the object to be reconstructed. As shown in Fig 3(b), based on the method of [11], we combine depth map data and MC method to extract the triangular mesh surface of the object from the visual hull. \(V_i\) is the reconstruction result of standard VbVH method, and is combined with the 2.5 dimensional depth map \(V_t\) to generate the more accurate reconstructed triangular surface \(V_r = V_i \cap V_t\). Specifically, for a voxel marked as Inside or Uncertain, we compute a difference value \(D_i\) between its distance to TOF camera \(A_i\) and its depth value \(E_i\) in depth map:

\[
D_i = A_i - E_i
\]

Then according to the difference values of all nodes of each voxel, we compute the positions of vertices of the extracted triangular mesh. For a voxel, if there exists \(D_i \cdot D_j < 0\) for node \(i\) and node \(j\), the position of the vertex generated by combining MC method and depth map data is:

\[
\text{Pos}_{ij} = \frac{P_i + P_j \times \text{scale}}{1 + \text{scale}},
\]

where \(P\) denotes the node’s position, \(\text{scale} = \left| \frac{D_i}{D_j} \right|\). For an arbitrary node \(k\) of a voxel, if \(D_k > 0\), we exploit the silhouette information to correct the surface mesh extracted by MC method. As shown in Fig 4, the projection \(w_1\) of node 1 on each imaging plane lies completely inside the silhouette, and there exists a projection \(w_2\) of node 2 on an imaging plane lying outside the silhouette. In this case, on the imaging plane \(plane_i\), we use the traversal algorithm like Bresenham algorithm [19] to calculate the intersection point \(S_i\) of the silhouette and the line \(w_1w_2\). By computing the back projection of \(S_i\), we obtain the vertex of the extracted triangular surface.
The polyhedral mesh extracted by MC algorithm needs to be textured for rendering. As shown in Fig 3(c), we adopt a view-dependent texture mapping algorithm which can obtain a good visual quality from the viewpoints of the cameras [14]. Take one triangular surface ΔABC for example, when rendering in current viewpoint View, we choose the closest camera direction as the texture source. In Fig 3(c), we can see the relationship between the desired viewpoint and the cameras’ directions. Suppose that there are \( N \) cameras. Obviously if the optical axis of the camera is the same with the viewpoint direction, the projection on this camera’s image plane is the most clear and suitable texture it can choose. Therefore, we just need to compute the angle between View and each optical axis of the camera \( Op_j \). Denote the angle between View and \( Op_j \) as \( \theta_j \), \( j = 1, 2, \cdots, N \). If \( \theta_1 \) is the smallest among \( \{\theta_j\} \), we project the three vertices of ΔABC on imaging plane \( I_1 \), and select the area surrounded by the projected triangle as the final texture.

### Virtual deformation simulation

We put forward an improved version of the hybrid deformation model [12, 20] to simulate the large deformation of virtual objects. As shown in Fig 4, this deformation model divides the body of deformable object into the geometrically-based deformation region (colored green) and the physically-based deformation region (colored blue). The geometrically-based deformation region which need not be physically accurate is simulated by FLSM method [21]. And we adopt the TLED method [22] to simulate the physically-based deformation region which is derived from continuum mechanics and thus is physically accurate. When we exploit the hybrid model to stably simulate the large deformation of virtual objects, the key is how to compute the deformation of calculation nodes from different deformation models, especially the interfacial calculation nodes (colored red) between two models. To achieve more stable deformation simulation, compared to the previous method [12, 20] which computes the deformations of FLSM, international and TLED nodes successively, our hybrid deformation model adopts a new method for computing three different nodes’ displacements in another order. Especially for the interfacial calculation nodes, we adopt a stable position-constrained method to compute their displacements, rather than the force-based method [12, 20] which needs to adjust a user-defined coefficient to get the desirable deformation. The calculating process of the improved hybrid model is detailed as follows:

#### Fig 4. Compute the intersection point of the silhouette and the projection line.

https://doi.org/10.1371/journal.pone.0214852.g004

...
Deformation computation of TLED nodes. As shown in Fig 5, the interfacial calculation nodes belong to the finite elements situated near the interface, so we compute their deformations using TLED method by treating them as the TLED nodes. We firstly calculate the deformation gradient matrix \( \mathbf{X} \) for each finite element using

\[
\mathbf{X}_{ij} = \frac{\partial x_i}{\partial x_j}, \tag{3}
\]

where 0 and \( t \) denote the reference and current attributes of nodes respectively, \( i \) and \( j \) denote the coordinate dimensional indices.

Then by multiplying a stationary strain-displacement matrix \( \mathbf{B}_L^{(a)} \) by \( \mathbf{X} \), we compute the complete strain-displacement matrix \( \mathbf{B}_L^{(a)} \mathbf{X} \) at any time \( t \)

\[
\mathbf{B}_L^{(a)} = \mathbf{B}_L^{(a)} \mathbf{X}, \tag{4}
\]

where \( a \) ranges from 1 to the nodes’ number per element, \( \mathbf{B}_L^{(a)} \) is defined in terms of the spatial derivatives of the element shape functions [22].

The second Piola-Kirchoff stress matrix of the finite element at the integration point is computed as

\[
\mathbf{S}_y = \mu (\delta_{ij} - \mathbf{C}_y^{-1}) + \lambda J (J - 1) \mathbf{C}_y^{-1}, \tag{5}
\]

where \( \delta_{ij} \) denotes Kronecker’s delta [22], \( \lambda \) and \( \mu \) denote Lame constants.

The nodal reaction forces of each element are computed by using Gaussian quadrature

\[
\mathbf{F} = \int \mathbf{B}_L^{(a)} \mathbf{S} \, \mathbf{dV}, \tag{6}
\]

where \( \mathbf{dV} \) denotes the element’s initial volume, and \( \mathbf{S} \) denotes the vector representation of the second Piola-Kirchoff stress.

Finally, for interfacial and TLED calculation nodes, we compute their net nodal reaction forces \( \mathbf{F} \) at time \( t \) by iteratively implementing (Eq 4) for each element. And then we explicitly
compute the displacement for each TLED node using central difference formula.

\[ \mathbf{u}_i^{(k+1)} = \frac{\Delta t^2}{M_k} (\mathbf{F}^{\text{external}} - \mathbf{F}_i^{(k)}) + 2\mathbf{u}_i^{(k)} - \mathbf{u}_i^{(k-1)} \]  

(7)

where \( M_k \) denotes the diagonal entry of the \( k \)th row of the diagonalized mass matrix, \( \mathbf{F}^{\text{external}} \) denotes the external nodal force.

**Deformation computation of interfacial and FLSM nodes.** The calculation nodes situated in the interface also belong to the regions of FLSM method. Our deformation model computes the deformations of interfacial and FLSM nodes by using the FLSM method. For each region \( R_i \) of the FLSM model, we compute the optimal rotation matrix \( \mathbf{R}_i \) and the translation vectors \( \mathbf{C}_0^i \) and \( \mathbf{C}_t^i \) by minimizing

\[ \sum_{i \in R_i} w_i (\mathbf{R}_i (\mathbf{x}_0^i - \mathbf{C}_0^i) + \mathbf{C}_t^i - \mathbf{x}_i^0), \]  

(8)

where \( w_i \) denotes the weight of individual lattice points. The optimal translation vectors are proved to be the mass center vector of the original shape and the mass center vector of the deformed shape, i.e.

\[ \mathbf{C}_0^i = \sum_{i \in R_i} w_i \mathbf{x}_0^i, \quad \mathbf{C}_t^i = \sum_{i \in R_i} w_i \mathbf{x}_t^i. \]  

(9)

For each region, we compute a matrix \( \mathbf{A}_i \) as

\[ \mathbf{A}_i = \sum_{i \in R_i} \mathbf{m}_i (\mathbf{x}_t^i - \mathbf{C}_0^i) (\mathbf{x}_0^i - \mathbf{C}_0^i)^T. \]  

(10)

We then compute the least squares rotation matrix \( \mathbf{R}_i \) for all particles of region \( R_i \) exploiting the matrix polar decomposition \( \mathbf{A}_i = \mathbf{R}_i \mathbf{U}_i \), where \( \mathbf{U}_i \) denotes a unique third order symmetric stretch matrix. For a region, the least squares rigid transformation of the rest position \( \mathbf{x}_0^i \) includes a rotation matrix \( \mathbf{R}_i \) and a translation that transfers the rotated vector \( \mathbf{C}_0^i \) to \( \mathbf{C}_t^i \). We use a vector \( \mathbf{T}_i \) to store this transformation.

\[ \mathbf{T}_i = \mathbf{R}_i (\mathbf{C}_0^i - \mathbf{R}_i \mathbf{C}_t^i). \]  

(11)

For each FLSM node or interfacial node, we compute a goal position vector \( \mathbf{g}_i \) which is the averaged least squares rigid transformation of the original position vector \( \mathbf{x}_0^i \) over the regions that the calculation node belongs to.

\[ \mathbf{g}_i = \frac{1}{|R_i|} \sum_{i \in R_i} \mathbf{T}_i \mathbf{x}_0^i. \]  

(12)

According to the FLSM method, for each FLSM node or interfacial node, we update the position vector \( \mathbf{x}_i \) and the velocity vector \( \mathbf{v}_i \) based on the goal position vector \( \mathbf{g}_i \):

\[ \mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \frac{\partial \mathbf{g}_i(t) - \mathbf{x}_i(t)}{\Delta t} + \frac{\mathbf{F}^{\text{external}}(t)}{m_i} \Delta t; \]  

(13)

\[ \mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \mathbf{v}_i(t + \Delta t) \Delta t, \]  

(14)

where \( \Delta t \) denotes the time step size.
Compared to our previous work [12, 20], we update the velocity vectors and position vectors of the interfacial calculation nodes by incorporating the reaction forces computed by (Eq 6) into the external forces of (Eq 13), which is a stable position-constrained method rather than a force-based method [12, 20] needing to adjust a user-defined coefficient.

**Deformation computation of embedded surface.** In our hybrid deformation model, both FLSM region and TLED region regularly place the calculation nodes in the form of axis-aligned cubes. The complex surface of object is embedded into the deformable cubes of the hybrid deformation model. For each vertex of the surface, the position vector \( x_i \) is calculated as the interpolation of the position vectors of the cube’s nodes.

\[
x_i = \sum_{j=1}^{n} \phi_j(x_i) x_j,
\]

where \( \phi(x) \) denotes the weight of each node of the cube.

**Virtual-reality interaction**

**Particle-based interaction simulation.** Based on our research on physically based fluid simulation [13, 14], we put forward a new particle-based algorithm for simulating one-way interaction of the reconstructed object and the virtual object, and two-way interaction of particle-based blood and hybrid model-based soft tissue. In our method, motivated by microscopic molecular dynamics, the calculation nodes of hybrid model are considered as Lagrangian particles of fluid, and the adaptively sampled vertices of the reconstructed object surface are considered as solid boundary particles of fluids. Thus, one-way interaction of the reconstructed object and the virtual object can be regarded as one-way fluid-solid interaction, which is directly solved by using multiphase weakly compressible smoothed particle hydrodynamics (WCSPH) [23] solvers.

First, we adopt the method of [24] to adaptively sample particles on the reconstructed object surface. Considering the non-homogeneous distribution of these sampled particles, we adopt the method of [13, 14, 24] to compute the density \( \rho_i \) of each particle \( i \) of the virtual object

\[
\rho_i = m \sum_j W(x_{ij}, h) + \sum_k \phi_{hk}(\rho_0) W(x_{ik}, h),
\]

where \( x_{ik} = x_i - x_k \), \( j \) is the index of neighboring particle of the virtual object, \( k \) is the index of neighboring particle of the reconstructed object, \( W \) is a Gaussian like kernel function with finite support radius \( h \), and \( \phi_{hk}(\rho_0) = \rho_0 V_{hk} \) scales the contribution of particle \( k \) to particle \( i \). \( V_{hk} \) is the current volume value of particle \( k \).

Then, we compute the interaction force \( F_{i \rightarrow k}^p \) exerted by particle \( k \) of the reconstructed object to particle \( i \) of the virtual object. The interaction force \( F_{i \rightarrow k}^p \) includes the normal force \( F_{i \rightarrow k}^{np} \) and the tangential force \( F_{i \rightarrow k}^{v} \), which can be computed by solving Navier-Stokes equations using WCSPH method

\[
F_{i \rightarrow k}^{np} = -m_i \phi_{hk}(\rho_0) \frac{p_i}{\rho_i^2} \frac{x_{ik}}{|x_{ik}|} W(x_{ik}, h),
\]

\[
F_{i \rightarrow k}^{v} = \mu m_i \phi_{hk}(\rho_0) \frac{v_{ik}}{\rho_i} \nabla W(x_{ik}, h),
\]

where \( \mu \) is surface friction coefficient, \( p \) is fluid pressure which is computed by Tait’s equation of [23], and \( v \) is velocity. In our framework, for a particle \( k \) of the reconstructed object, its
velocity $v_k$ is computed as

$$v_k = \frac{x^{\text{cur}} - x^{\text{pre}}}{\Delta t},$$  \hspace{1cm} (19)\]

where $x^{\text{cur}}$ is the centre of gravity of the current reconstructed object, and $x^{\text{pre}}$ is the recorded centre of gravity of the reconstructed object in the previous frame.

However, when the velocity difference becomes relatively larger, only relying on the above interaction forces results in the virtual object penetrating into the surface of the reconstructed object. To solve these problems, we incorporate the velocity-position correction scheme of [13] and [14] into our multi-GPU virtual-reality interaction framework.

For a particle $i$ of the virtual object and a particle $j$ of the reconstructed object, if there exists $v_{ij} \cdot n_j < 0$ and $|x_{ij}| < d$, the reconstructed object is considered to be penetrated by the virtual object at the position $x_j$. $d$ is a user-defined value, and $n_j$ denotes the surface normal of the reconstructed object at $x_j$. Fig 6 shows that a particle $i$ could penetrate more than one particle of the reconstructed object at the same time. In this case, we dynamically create a virtual boundary particle $k$ for the reconstructed object to collide with particle $i$. For this virtual particle $k$, its field quantity $A_k$, such as normal, position and velocity, is computed as the weighted average of the corresponding values of the neighboring penetrated particles:

$$A_k = \frac{\sum m_j A_j W(x_{ij}, d)}{\sum m_j W(x_{ij}, d)}.$$ \hspace{1cm} (20)

After the time integration of the deformation simulation, we first correct the position of $i$ along the normal of $k$

$$\tilde{x}_i = x_i + dn_k,$$ \hspace{1cm} (21)

where $\tilde{x}_i$ is the corrected position of $i$.
We then correct the velocity of $i$ based on boundary material property and momentum conservation law. The velocities of $i$ and $k$ are projected onto both the normal direction and the tangential direction of $k$, so we get four velocity components: $v^i_n$, $v^i_t$, $v^k_n$, and $v^k_t$. By enforcing momentum conservation constraint along these two directions, we obtain the following formulas:

\begin{align}
 m_i v^i_n + m_k v^k_n &= m_i \tilde{v}^i_n + m_k \tilde{v}^k_n, \quad (22) \\
 m_i v^i_t + m_k v^k_t &= m_i \tilde{v}^i_t + m_k \tilde{v}^k_t, \quad (23)
\end{align}

where $\tilde{v}^i_n$, $\tilde{v}^k_n$, $\tilde{v}^i_t$, and $\tilde{v}^k_t$ are corrected velocities.

Because the position and velocity of the reconstructed object only depend on the real object, the virtual-reality coupling is one-way. Thus we set the corrected velocities of particle $k$:

\begin{align*}
 \tilde{v}^k_n &= (0, 0, 0)^T, \\
 \tilde{v}^k_t &= (0, 0, 0)^T.
\end{align*}

Substituting it into Eqs (22) and (23), we get

\begin{align}
 \tilde{v}^i_n &= \frac{m_i v^i_n + m_k v^k_n}{m_i}, \quad (24) \\
 \tilde{v}^i_t &= \frac{m_i v^i_t + m_k v^k_t}{m_i}. \quad (25)
\end{align}

As for two-way coupling of particle-based blood and hybrid model-based soft tissue, we also use the above coupling method. The differences are the nodes of hybrid model-based soft tissue have their own velocities, and the corrected velocities for the virtual particle are set according to the method of [13].

**Virtual cutting simulation.** Virtual cutting of deformable objects is one of the important 3D interactive manipulations in virtual-reality interaction environments. Based on our previous work [12, 20], by combining the improved hybrid deformation model and the virtual node algorithm [25], we propose to use the scalpel reconstructed from real images to cut the virtual deformable objects. In our virtual-reality interaction framework, the cut operation of the reconstructed scalpel is restricted to the region simulated by TLED method. The cutting procedure includes 4 steps:

Step 1. Finding the elements which are completely cut

Firstly, it implements the collision detection procedure to find the elements which intersect with the cutting plane swept by the reconstructed scalpel. Spatial hashing algorithm is adopted to accelerate time-consuming collision detection. As shown in Fig 7, after collision detection, we find that three elements intersect with the cutting plane: $abij$, $bchi$, and $cdgh$.

![Fig 7. Finding the completely cut elements.](https://doi.org/10.1371/journal.pone.0214852.g007)
Among these three elements, our method regards \textit{abij} and \textit{bchi} as the completely cut elements, because the graph structures formed by their 4 nodes are unconnected. For the completely cut elements, our method allocates two arrays to separately store the information of nodes located in each side of the cutting plane.

Step 2. Generating the virtual nodes

Fig 8 shows the principle of generating virtual nodes for the completely cut elements. For each node of the completely cut elements, if all its attached elements are completely cut elements, our method generates a virtual node for it. As shown in Fig 7, for \(a, b, i\) and \(j\), our method generates \(a_0, b_0, i_0\) and \(j_0\) as their virtual node respectively.

Step 3. Allocating the virtual nodes

When the virtual nodes are generated, our method allocates these nodes in the arrays of the completely cut elements. As shown in Fig 9, for the completely cut element \textit{abij}, the node \(a\) stored in the first array has a virtual node \(a'\), so we fill the virtual node \(a'\) into the corresponding position of the second array. As for the node \(c\), because it does not has the virtual node, we fill it into the second array of the element \textit{bchi}.

Step 4. Creating new generated elements

As shown in Fig 10, for each array of the completely cut elements, our method creates a new element according to the node index. Then the pre-computed quantities are copied from the completely cut element to the newly generated elements. Each new element contains a part of material of the completely cut element, such as mass. For each new element, by only rendering the part situated at the side of the cutting plane, our method produces very smooth surface of the incision.
Multi-GPU implementation

The proposed virtual-reality interaction simulation framework is entirely implemented on the multi-GPU platform by using CUDA for the real-time performance. As the memory between GPUs is not shared, parallelism across multiple GPUs would be less fine grained than that on a single GPU to reduce the amount of interaction and number of memory copies between each GPU. As shown in Fig 11, our multi-GPU implementation method respectively splits the workload of 3D reconstruction, hybrid deformation and fluid simulation into many smaller parts that can be computed respectively on different GPUs. The number of subtasks equals the number of GPUs. The halo of the subspace is represented by the colored edges in Fig 10. Each subspace holds a copy of the edge of its adjacent subspaces to take fast neighbor access into account. Because the neighbor information of each computation node is not changed during the simulation, the computation spaces of 3D reconstruction and hybrid deformation are evenly split into subspaces and then distributed to each GPU. As for SPH fluid simulation, the particles freely moving in space require the solver to frequently update the corresponding workload of the respective GPUs. In this paper, we adopt the sophisticated load balancing strategy of [26] to achieve optimal performance, which frequently implements a domain
decomposition correction to adjust the current distribution of the workloads among the GPUs.

Multi-GPU implementation using CUDA is especially suitable for addressing the problems that can be indicated as data-parallel computations, so during pre-processing phase, the data elements should be made parallel and independent. The reconstruction space is divided into uniform and independent voxels with certain resolution. Then projections of each voxel onto every image plane are calculated according to the calibration information. Because the data access is stochastic during execution, the data should be stored into texture memory of GPU rather than global memory which gets maximum memory bandwidth only when following coalesced access pattern. Other information should also be stored into texture memory, such as look-up tables of marching cube algorithm, binary image of received foreground silhouette, and pre-computed data for hybrid deformation model.

The intersection test of voxels’ projections and foreground silhouettes is an important content for visual hull computation. We define a kernel named classifyVoxels() executed for each voxel, and obtain a voxel state sequence shown in Fig 12(a). The voxels marked as Inside and Outside have no intersections with the reconstructed objects’ surface, so voxels of these two types are labeled as 0, the Uncertain voxels having intersections with the objects’ surface are labeled as 1. So as shown in Fig 12(b), the voxels’ state sequence is binarized parallelly, thus it is composed of digits 0 and 1. The triangularized isosurface should be extracted from the Uncertain voxels which are set to 1 in the binarized state sequence, but in fact most of the voxels are set to 0. Thus, to improve the performance, we must compress the voxels’ status sequence and remove the digits 0. Our method uses the cudppScan() function of CUDPP library to implement parallel prefix sum on GPU and get the scan sequence of the voxels’ status sequence (Fig 12(c)). Then we launch compactVoxels() kernel to implement parallel reduction, and gets the Uncertain voxel sequence (Fig 12(d)).

Based on NVIDIA CUDA ‘Marching Cubes Isosurfaces’ demo, the exact intersection points on the edges of the Uncertain voxels are computed, and then a triangular mesh surface is extracted by connecting these intersection points. The computations are implemented in parallel on GPU by launching the generateTriangles() kernel. During parallel triangulation, the buffer object VBO is registered and mapped by using the mutual-operation property of CUDA and OpenGL. When the position vector, normal vector and texture coordinate of each triangular vertex are computed by using CUDA, these data are transferred to OpenGL through VBO.
Efficient texture mapping improves the visual quality of the reconstructed objects. In image-based modeling system, the textures derive from the foregrounds of images captured by different cameras, so each triangle has many possible mapped textures. We must store all textures and choose the most suitable one. Based on Multitexture Mapping and programmability of current GPU, we present a new texture mapping method applied to image-based modeling system. Firstly, the method enables the Multitexture Mapping and sends the normals and texture coordinates of each triangular vertices stored in buffer objects. Then the vertex shader programmed by cg language outputs the normals and texture coordinates. The pixel shader chooses the suitable texture of each fragment according to the angles between the normal and every optical axis of the cameras. This shader reads the texture from texture sampler variables which must be provided by the application as uniform parameters to a Cg program.

To simulate virtual-reality interaction, the search of neighboring particles for each particle is required in each time step, which is the most time-consuming component. In order to accelerate this computation, based on the uniform spatial hashing algorithm using Z-indexing [27], we propose a new GPU-based neighboring particle search method. We find that the order of the particles in a grid cell does not affect neighboring particle search. As shown in Fig 13, different from the method of [27], our method only implements once Z-indexing sort operation for grid cells, but not for particles to avoid additional sorting: First, we parallelly compute the number of particles in each grid cell. Then, our method implements parallel prefix sum on GPU and sorts for particle array. For example, the grid cell with an index of 211 has 4 particles, the grid cell with an index of 212 has 3 particles, and so on. Finally, we implement radix sorting for grid cells based on Z-indexing. Each particle has an offset value relative to the starting position in the sorted grid index array, and thus we can parallelly insert the particle into the sorted array avoiding access conflicts.

Results and analysis
We have tested the multi-GPU accelerated virtual-reality interaction simulation method via several example scenarios. The hardware and software configuration of our multi-GPU accelerated virtual-reality interaction framework is shown in Table 1.
Reconstruction and interaction results

Fig 14 shows the reconstructed results of two real objects placed in the 1m × 1m × 1m sized frame: a toy bunny (Top) and a real hand (Bottom). From left to right: the captured image, the triangle mesh surface of the reconstructed object and the reconstructed object surface with texture mapping. In this example, the input image resolution is 640 × 480, and the division resolution of the real space is 64 × 64 × 64. The frame rate can be achieved 35.0 FPS.

Fig 15 shows the comparison results of the proposed reconstruction method and the previous VBVH reconstruction methods. From left to right: the reconstruction result of Kim’s method [28], the reconstruction result of Shao’s method [14], and the reconstruction result of our VBVH method by using our 3m × 3m × 3m sized frame. Kim’s method produces smooth surface by using multiple view stereo matching algorithm, but many vertices are mistakenly

| Item                | Description                                                                                   | Number |
|---------------------|----------------------------------------------------------------------------------------------|--------|
| Client              | Intel Core 2 Duo E6400 2.13 GHz, 4GB DDR2-667 memory, WIN7 PRO 64. IDE: Visual Studio 2013, OpenCV 2.0. | 8      |
| Server              | Intel Core 2 Duo E6400 3.5 GHz, 16GB DDR2-667 memory, WIN7 PRO 64, 6x NVIDIA GeForce GTX 650. IDE: Visual Studio 2013, OpenGL 4.0, CUDA 7.0, Cg 3.0. | 1      |
| Optical Camera      | PointGrey Fleex2, resolution 640x480, 30fps                                                  | 8      |
| TOF Depth Camera    | PMD CamCube 2.0, resolution 200x200, 30fps                                                   | 1      |
| Network             | 1Gbit Ethernet                                                                               | /      |

https://doi.org/10.1371/journal.pone.0214852.t001

Table 1. Experimental environment.

Reconstruction and interaction results

Fig 14 shows the reconstructed results of two real objects placed in the 1m × 1m × 1m sized frame: a toy bunny (Top) and a real hand (Bottom). From left to right: the captured image, the triangle mesh surface of the reconstructed object and the reconstructed object surface with texture mapping. In this example, the input image resolution is 640 × 480, and the division resolution of the real space is 64 × 64 × 64. The frame rate can be achieved 35.0 FPS.

Fig 15 shows the comparison results of the proposed reconstruction method and the previous VBVH reconstruction methods. From left to right: the reconstruction result of Kim’s method [28], the reconstruction result of Shao’s method [14], and the reconstruction result of our VBVH method by using our 3m × 3m × 3m sized frame. Kim’s method produces smooth surface by using multiple view stereo matching algorithm, but many vertices are mistakenly
deleted. Shao’s method generates many reconstructed noises which reduces the smoothness of the surface. In contrast, by exploiting the data of TOF depth camera to correct the triangulated surface generated by MC algorithm, our extended VbVH method can produce more smooth and accurate reconstruction results. Compared to Zuo’s method [29], our reconstruction method can not handle concave and non-Lambertian objects, but can meet real-time performance requirements by using multiple GPUs.

Fig 16 shows the deformation results of the bunny modeled by our improved hybrid deformation method. The ratio of TLED nodes and LSM nodes is 6:4. From the animation results, our method could smoothly model the larger deformations of the object divided into two types of regions.

Fig 17 shows two virtual-reality interaction examples for the 1m×1m×1m sized frame. For the first example, we let the reconstructed toy bunny to collide with the virtual bunny animated by the improved hybrid model. The second example shows the reconstructed hand touches the virtual bunny. The results demonstrate that our method can effectively simulate virtual-reality interaction. Especially, the penetration artefacts are avoided in the interaction simulation.

Fig 18 is a scenario that the reconstructed scalpel cuts the virtual liver fixed at one end. From the results, our improved hybrid deformation method could stably model the larger
deformations of virtual soft tissues, and our cutting method could create smooth cutting incisions. The division resolution of the real space is $64 \times 64 \times 64$, and the ratio of TLED nodes and LSM nodes is 7:3. In the simulation of virtual cutting, no texture mapping is used for the surface of the virtual object.

Fig 19 shows that the reconstructed scalpel cuts the virtual liver with the simulation of bleed effects. From the results, we can conclude that our virtual cutting method can produce smooth incisions, and our virtual-reality interaction method can realistically and stably simulate one-way coupling of the reconstructed object and the virtual object, and two-way coupling of particle-based blood and hybrid model-based soft tissue. The division resolution of the real space is $64 \times 64 \times 64$, and the maximum number of SPH particles is 40k. Compared to the virtual cutting simulator using the haptic feedback devices [30], the obvious limitation of our framework is that the cutting plane cannot be accurately calculated from the reconstructed scalpel, but the virtual-reality interaction model can be applied to the remote virtual surgery.

**Time performance analysis**

The division resolution of the real space is a critical influence factor of the time performance of the VbVH based reconstruction process. Table 2 shows the comparisons of time performance between CPU and multi-GPU implementation of the reconstruction process in different division resolutions. From the results, our multi-GPU based VbVH reconstruction
method can obtain real-time performance for moderately complicated computations. Compared to the CPU implementation method, our multi-GPU implementation method achieves impressive speedups of about 14 times for the division resolution 100*100*100. And compared to the single-GPU implementation method, our multi-GPU implementation method achieves speedups of about 3.5 times.

For the deformation simulation, our improved hybrid method could simultaneously maintain the physical accuracy of TLED and the computational efficiency of FLSM. We could regulate the balance between computational efficiency and physical accuracy through dynamically adjusting the ratio of FLSM nodes and TLED nodes. Fig 20 shows the time performance of the

| 3D reconstruction | 64’64’64 | 100’100’100 |
|-------------------|---------|-------------|
|                   | Hand    | Bunny       | Hand    | Bunny       |
| CPU version       | 180.4   | 194.6       | 554.5   | 596.8       |
| Single-GPU version| 57.6    | 64.4        | 137.5   | 149.8       |
| Multi-GPU version | 26.2    | 27.5        | 39.3    | 41.5        |

Fig 20. Time performance of the improved hybrid deformation model.

hybrid deformation model using different ratios of TLED nodes and FLSM nodes. Ratio_1 is set to 7:3, and Ratio_2 is set to 3:7.

Our multi-GPU implementation method can significantly improve the time performance of each part of the proposed virtual-reality interaction framework. Table 3 shows that the multi-GPU implementation of our texture mapping method is much faster than the CPU implementation and the single-GPU implementation. The maximum acceleration ratios are 7.5 and 1.8 respectively for the division resolution 64×64×64. From the results shown in Table 4, compared to the GPU implementation of Z-index sorting method [27] and uniform space partition method [13], our new multi-GPU based method significantly improves the time performance of neighboring particle search.

In Table 5, we demonstrate the time performance comparison of CPU based and multi-GPU based virtual-reality interaction framework. From the results, our multi-GPU implementation method of the whole virtual-reality interaction framework significantly improves the time performance. The maximum acceleration ratio is 14.5. And compared to the single-GPU based virtual-reality interaction framework of [14], our method also has obvious performance advantages. Under the moderate calculation scale, the frame rate of our method can be achieved about 25 FPS, which is suitable for real-time virtual reality applications.

**Conclusion**

We have designed a real-time multi-GPU accelerated virtual-reality interaction simulation framework where the reconstructed objects from camera images interact with virtual deformable objects. It glues image-based 3D reconstruction, hybrid model based deformation, virtual cutting, particle-based coupling and parallel computation on multiple GPUs for a new
immersive interactive experience. The experiment results have demonstrated that our multi-GPU accelerated virtual-reality interaction framework achieves real-time performance under the moderate calculation scale, which provides a new effective 3D interaction technique for virtual reality applications.

In future, our efforts are geared towards simulating more types of virtual-reality interactions and determining more sophisticated load balancing strategy for multi-GPU architecture.

Supporting information

S1 File. 3D reconstruction experiment code. (ZIP)

S2 File. Virtual cutting experiment code. (ZIP)

Author Contributions

Conceptualization: Xuqiang Shao.

Data curation: Fengquan Zhang.

Funding acquisition: Weifeng Xu.

Resources: Weifeng Xu.

Software: Weifeng Xu, Lina Lin.

Validation: Weifeng Xu, Lina Lin.

Visualization: Lina Lin.

Writing – original draft: Xuqiang Shao.

Writing – review & editing: Fengquan Zhang.

References

1. Bro-Nielsen M, Helfrick D, Glass B, Zeng X, Connacher H. VR simulation of abdominal trauma surgery. Studies in Health Technology and Informatics. 1998; 50: 117–123. PMID: 10180526

2. Webster RW, Zimmerman DI, Mohler BJ, Melkonian MG, Haluck RS. A prototype haptic suturing simulator. Studies in Health Technology and Informatics. 2001; 81: 567–569. PMID: 11317811

3. Bielser D, Gross MH. Open surgery simulation. Studies in Health Technology and Informatics. 2002; 85: 57–63. PMID: 15458060

4. Rianto S, Li L. Fluid dynamic visualisations of cuttings-bleeding for virtual reality heart beating surgery simulation. Proceedings of the 33rd Australasian Conference on Computer Science. Darlinghurst, Australia: Australian Computer Society. 2010; 102: 53–60.

5. Narayanan PJ, Rander PW, Kanade T. Constructing virtual worlds using dense stereo. Proceedings of International Conference on Computer Vision. Los Alamitos, IEEE Computer Society Press. 1998; 3.

6. Kanade T, Rander P, Narayanan PJ. Virtualized reality: constructing virtual worlds from real scenes. IEEE Multimedia. 1997; 4(1): 34–47.

7. Yuan J, Wang Q, Li B. Using multi-view network constraints among reference marker points to realize coarse registration in structured light system with high accuracy. Journal of Computer-Aided Design & Computer Graphics. 2015; 27(4): 674–683.

8. Lazebnik S, Furukawa Y, Ponce J. Projective Visual Hulls. International Journal of Computer Vision. 2007; 74(2): 137–165.

9. Chen L, Kwan-Yee KW. Exact Visual Hull from Marching Cubes. Proceedings of VISAPP08. 2008; 2: 597–604.

10. Allard J, Menier C, Raffin B, Boyer E, Faure F. Grimage: Markerless 3D Interactions. Proceedings of ACM SIGGRAPH, New York, NY, USA. 2007: 1–9.
11. Wang C, Zhou Z, Wu W. Real-time 3D reconstruction using time of flight sensor optimized visual hull. Journal of Computer-Aided Design & Computer Graphics. 2005; 17(12): 494–501.
12. Shao X, Zhou Z, Wu W. A Hybrid Deformation Model for Virtual Cutting. Proceedings of IEEE International Symposium on Multimedia, Taichung, Taiwan. 2010: 1–8.
13. Shao X, Zhou Z, Magnenat-Thalmann N, Wu W. Stable and fast fluid-solid coupling for incompressible SPH. Computer Graphics Forum. 2015; 34(1): 191–204.
14. Shao X, Nie X, Wang B. GPU-based real-time 3D visual hull reconstruction and virtual-reality interaction. Journal of Computer-Aided Design & Computer Graphics. 2017; 29(1): 52–61.
15. Zhang S, Wang C, Shao X, Wu W. DreamWorld: CUDA-accelerated real-time 3D modeling system. Proceedings of Virtual Environments, Human-Computer Interfaces and Measurements Systems. 2009: 1–8.
16. Zhang Z. Camera Calibration with One-Dimensional Object. IEEE transactions on pattern analysis and machine intelligence. 2004; 26(7): 892–899. https://doi.org/10.1109/TPAMI.2004.21 PMID: 18579947
17. Elgammal A, Harwood D, Davis L. Non-parametric model for background subtraction. Proceedings of European Conference on Computer Vision, Berlin, Heidelberg, Springer, 2000; 1843.
18. Lorensen WE, Cline HE. Marching cubes: A high resolution 3d surface construction algorithm. Proceedings of the 14th annual conference on Computer graphics and interactive techniques, Anaheim, California, USA, 1987: 163–169.
19. Wright WE. Parallelization of Bresenham’s line and circle algorithms. IEEE Computer Graphics and Applications. 1990; 10(5): 60–67.
20. Shao X, Zhao W. Multi-precision hybrid model for elastic deformation and interaction simulation. Journal of Computer-Aided Design & Computer Graphics. 2016; 28(6): 960–967.
21. Rivers AR, James DL. FastLSM: fast lattice shape matching for robust real-time deformation. ACM Transactions on Graphics. 2007; 26(3): 82:1–82:10.
22. Miller K, Joldes G, Lance D, Wittek A. Total Lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. Communications in Numerical Methods in Engineering. 2007; 23 (2): 121–134.
23. Becker M, Teschner M. Weakly compressible SPH for free surface flows. Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Aire-la-Ville: Eurographics Association Press, 2007: 209–217.
24. Akinci N, Comelis J, Akinci G, Teschner M. Coupling elastic solids with SPH fluids. Computer Animation and Virtual Worlds. 2013; 24(3–4): 195–203.
25. Molino N, Bao Z, Fedkiw R. A virtual node algorithm for changing mesh topology during simulation. Proceeding of ACM SIGGRAPH. New York, NY, USA, 2004: 385–392.
26. Verma K, Szewc K, Wille R. Advanced load balancing for SPH simulations on multi-GPU architectures. Proceeding of IEEE High Performance Extreme Computing Conference, Waltham, MA, USA, 2017.
27. Goswami P, Schlegel P, Solenthaler B, Pajarola R. Interactive SPH simulation and rendering on the GPU. Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 2010: 55–64.
28. Kim YM, Theobalt C, Diebel J, Kosecka J, Miscusik B, Thrun S. Multi-view image and TOF sensor fusion for dense 3D reconstruction. Proceedings of International Conference on Computer Vision Workshops. 2009: 1542–1549.
29. Zuo X, Du C, Wang S, Zheng J, Yang R. Interactive visual hull refinement for specular and transparent object surface reconstruction. Proceedings of IEEE International Conference on Computer Vision. 2015: 2237–2245.
30. Cheng Q, Liu P, Lai P, Xu S, Zou Y. A novel haptic interactive approach to simulation of surgery cutting based on mesh and meshless models. Journal of Healthcare Engineering. 2018: 1–16.