Tangible Images of Real Life Scenes

Xingzi Zhang[a], Michael Goesele[b], Alexei Sourin[a]

[a] *School of Computer Science and Engineering, Nanyang Technological University, Singapore*
[b] *TU Darmstadt, Germany*

Abstract

Haptic technologies allow for adding a new "touching" modality into virtual scenes. However, 3D reconstruction of real life scene often results in millions of polygons which cannot be simultaneously visualized and haptically rendered. In this paper, we propose a way of haptic interaction with the reconstructed real life scenes where multiple original images of the real scenes are augmented with the reconstructed polygon meshes. We present our solution to the problems of haptic model alignment with the images and interactive haptic rendering of large polygon meshes with reconstruction artifacts. In particular, the presented collision detection algorithm is not restricted by any hypothesis and robust enough to support smooth interaction with millions of polygons. The feasibility and usability of the proposed solution is evaluated in a user study.

Keywords:
haptic interaction; tangible image; large-scale imperfect polygon mesh

1. Introduction

Haptic technology, or haptics, is an interaction feedback technology based on applying forces, vibrations, and/or motions to the user. Usually, haptic interaction is considered with 3D objects defined by polygons. However, 3D reconstruction of a real life scene using computer vision techniques often results in millions of polygons which cannot be simultaneously visualized and haptically rendered. Mesh simplification methods and acceleration techniques can help, however in many cases the visual display of a photorealistic scene still creates a very significant and time consuming overhead to the whole project implementation pipeline. Replacement of the actual 3D scenes with their images is actively used in image-driven visualization such as interactive panoramas, street walkthroughs, and online shopping with interactive images. Similarly, replacement of the interactive 3D scenes with their "tangible images" is an alternative solution to this problem.

Haptic interaction with images, as if they were actual 3D scenes, can be done in a few different ways, which were also previously explored: Firstly, the haptic forces can be derived directly from the image by analyzing pixel intensity [1]. This approach, however, imposes restrictions on the scene illumination. Secondly, haptic components can be added to the images and used for haptic interaction by sketching simplified haptic models on the image so that the models were eventually matched with the respective parts of the displayed scene [2]. Thirdly, in case when there are available reconstructed polygon meshes, they can be also matched with the image and only used for haptic interaction while the original image is displayed thus liberating the computer from 3D visualization task. We proposed our initial solution to this problem in [3] where we mostly worked on the haptic rendering algorithm for large and imperfect polygon meshes.

In this paper we continue this research solving a problem of haptic interaction with the reconstructed real life scenes where *multiple original images of the real scenes* are augmented with the reconstructed polygon meshes. This required us to solve problems of haptic model alignment with multiple images to be displayed as well as smooth interactive haptic rendering of large multi-million polygon meshes, which may have inevitable reconstruction artifacts.

In Section II, we survey the relevant works. In Section III, we discuss the overall project pipeline, describing how to match the reconstructed mesh with the image and how to perform haptic interaction with large-scale imperfect meshes. Results of the proposed algorithm are provided in Section IV. The design and evaluation of the usability test is presented in Section V to prove the feasibility and usefulness of the presented tangible images approach, followed by the conclusion in Section VI.

2. Related Work

2.1. Visual Rendering in a Visual-haptic Interaction Environment

In a visual-haptic interactive scene, polygon meshes, as well as the haptic cursor, are usually displayed for visual feedback. Haptic rendering on large meshes is discussed in section 2.3. In this section we talk about the problem in visual rendering, which is that even if the large mesh can be

Email addresses: ZHAN0388@e.ntu.edu.sg (Xingzi Zhang), goesele@cs.tu-darmstadt.de (Michael Goesele), assourin@ntu.edu.sg (Alexei Sourin)

haptically and visually rendered, displaying haptic cursor along with the mesh is problematic. The reason is given in the next paragraph.

The haptic cursor position is computed by the CPU (together with other haptic rendering tasks) at the rate of 1 kHz. In each graphics frame (30-60 Hz), the cursor position is read from the haptic callback function for visual display of the cursor. Thus samples of cursor position are displayed at graphics update rate. As we know, the graphics rendering time increases with increasing mesh size. This would in turn lead to an increase in sampling interval of cursor position (as in Fig. 1), resulting in clumsiness in the displayed cursor movement. To reduce the graphics rendering time for visual models, we need to either speed up the rendering process or to reduce the size of the models.

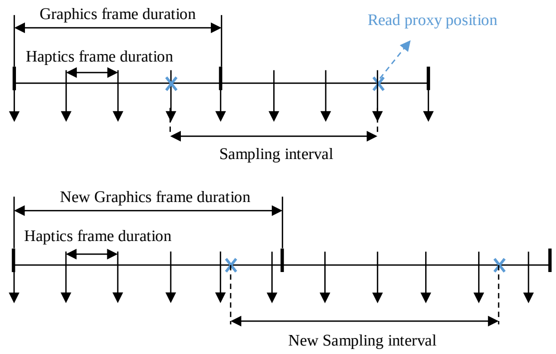


Fig. 1. Illustration of the effect of increasing graphics rendering time in one frame. Sampling interval is always equal to the graphics frame duration. We need to keep the sampling interval small in order to display the haptic cursor consistently.

Common graphical renderers in visual-haptic interaction, such as OpenGL and Direct3D, utilize rasterization-based rendering due to the real-time requirement. With powerful graphics hardware and the use of acceleration structures for culling, a complex interactive scene can be rendered in real-time. However, the level of realism of the rendered scene heavily depends on the lighting techniques applied to the scene and the manual efforts of designers, which poses an obstacle to realistic immersion. Compared to rasterization-based algorithms, ray tracing provides a more realistic visual effect, but it is costly in computation. With the emergence of high-performance rendering engines like Brigade [4], it has become possible to incorporate ray tracing into real-time rendering. However, it is still far from being applied in interactive visual-haptic scenes with millions of polygons.

In order to display a more realistic scene, there are works combining ray tracing with rasterization-based rendering in a visual-haptic interaction environment. For example, Morris and Joshi propose to display pre-processed raytraced images to simulate a static-viewpoint scene [5]. Depth information is extracted here along with the image for proper occlusion with other objects rendered in real-time. In this way, costly computation is avoided in the rendering loop and visual realism is improved.

Based on previous work [5], we know that images can be a promising alternative to displaying the models in some real-time applications. For real life scenes, images provide high-resolution visual feedback without complex computations.

2.2. Haptic Interaction with Images

Methods for haptic interaction with images can be roughly categorized into two groups. The first group of methods generate force feedback based on image processing techniques. They first build a correspondence between the derived image properties (e.g., grayscale or color values of the pixels) and the model (e.g., depth map or 2.5D geometry model) for force calculation, and then compute the force on-the-fly. These methods allow us to feel the object edges and textures as well as its visible geometry in certain cases. The whole image scene is, however, perceived tangibly only as an embossment of the relief. Besides, since none of the properties can always represent the actual scene geometry of any image, these techniques can only be applied to a specific group of images (e.g., frontally illuminated images).

The second group of methods augments images with haptic models matching the image content. In this way, the users are allowed to perceive the full 3D geometry of the objects in the images, including the invisible surfaces. The augmented models can be geometry models, depth maps, or even mathematical functions and procedures [6-7].

High requirement for haptic refresh rate, however, imposes a constraint on the computation time. If we want to use polygon model in the interaction, we need to make a tradeoff between the complexity of the polygon model and the continuity of the force feedback. Some methods use simplified meshes to meet the real-time requirement and provide additional information to simulate haptic details on the surface. For example, M. A. Otaduy et al. [6] extract 3D texture-induced force from texture images and apply it along with low-resolution geometry-induced force. Kim et al. [8] propose to define geometric information as a depth map while stiffness and viscosity maps are applied at the same time to represent physical properties of the scene. To avoid the constraint, there are also methods that resort to other geometry representation. In our previous paper [2], we define the basic geometry of the models using FRep models (variants of implicit functions) and add texture force to simulate details. All these methods allow the users to perceive the haptic details to some extent, but none of them manage to tangibly present the high-fidelity geometry of the objects in real life images.

Multi-view reconstruction methods such as MVE [9] are able to produce polygon meshes of a complex scene which are sufficient for visualization. If we use the reconstructed model to provide haptic feedback for the images served as input in the reconstruction pipeline, the haptic display could be easily registered with the images. In this way, a high-resolution haptic feedback can be achieved we are able to deliver a realistic haptic immersion into images as if they were 3D scenes. Therefore, if we could find a way to handle collision detection with reconstructed meshes, a reconstructed model is an ideal choice for the haptic interaction with images.

2.3. Point-based Haptic Rendering with Polygon Meshes

The challenge of collision detection with large-scale meshes lies in locating the polygon that the haptic cursor (Haptic Interface Point, HIP) is in contact with in real-time. We call it the *active polygon* in this paper. In a virtual scene, a *proxy* is calculated to indicate the position of HIP. When the HIP moves in the free space, the proxy position matches with the position of the HIP. When the HIP collides with the mesh, i.e. inside the mesh, if it is a simulation of rigid-to-rigid collision, the proxy lies on the surface of the active polygon.

Many existing methods for haptic rendering of polygon meshes detect collision with the whole polygon mesh in each haptic frame. The haptic rendering time thus depends on the number of polygons. For example, in widely-used haptic rendering methods such as God-Object [10], Ruspini [11] and CHAI3D [12], active constraint polygons need to be found first from all the polygons in each haptic frame, and then the constraint polygon with the shortest distance to the haptic cursor is determined as the active polygon. OpenHaptics HLAPI [13] utilizes the OpenGL Depth Buffer and Feedback Buffer to access shapes rendered in graphics rendering loop and automatically detect collison based on the geometry and depth information stored inside these two buffers. In this way, HLAPI's performance is not influenced by the size of polygons. However, the Feedback Buffer has a limited size (storing up to 65536 vertices) and using the Depth Buffer results in discontinuities in the computed haptic force due to the fact that 3D geometry is saved as an image in the Depth Buffer.

There are a number of methods that have been proposed to reduce the computational time using spatial partitioning and hierarchical structures, such as H-COLLIDE [14] and ActivePolygon [15]. In the ActivePolygon algorithm, polygons are stored in an octree data structure. Only the polygons stored in the cells that the haptic cursor passes by between frames are used for collision detection. These methods could effectively reduce the haptic rendering time, however, they cannot handle the situation when the mesh is too dense, because the computation complexity of these algorithms depends on the number of polygons in the cells that the haptic cursor passes from frame-to-frame. Thus, if the haptic cursor moves very fast and passes several cells within one cycle, only the first cell (obtained from the cursor position in last frame) and the last cell (obtained from the cursor position in current frame) are known while the in-between cell information is lost. To avoid missing the active polygon, all the cells that the cursor might pass need to be considered and this would lead to a significant expansion in the search range, even if the cell size is optimized. For example, the maximum velocity of the Geomagic Touch desktop haptic device is 2.5 mm/ms, so all the polygons in those cells within the distance of 2.5 mm to the previous position of haptic cursor need to be checked. If the mesh is dense and has a few hundred polygons within a 2.5 mm cubic space, fast and accurate collision detection cannot be maintained.

Geometry connectivity information was first used by Chih-Hao Ho et al. in their "neighborhood watch" algorithm [16] to predict the next active primitive (an extension of active polygon) based on the previous active primitive. It refers to the vertex, line segment or polygon that the haptic cursor is in contact with. Before haptic rendering, the connectivities among vertices, lines and polygons of the mesh are predefined and stored. After the first collision is detected, only the neighbors of the previous active primitive are checked. Using an iterative approach one can track the trace of the haptic cursor and find the closest primitive at the current position. In this way, the haptic rendering time is independent of the number of polygons except for every first collision with the mesh.

Inspired by the "neighborhood watch" algorithm [16], we propose a hybrid collision detection method which combines the pre-computed connectivity information and spatial partitioning. Instead of directly searching for the active primitive, we first track the polygon intersected with HIP trace and then use it as start point to track the active primitive. In this way, the computational time is fully independent of the polygon number. One of the main differences between our proposed method and Chih-Hao Ho's method is that we are not dealing with perfect CAD polygon meshes. The geometry information obtained from the meshes can be incomplete, may contain redundant vertices and facets, or may even be wrong. Thus more general criteria for searching for the active primitive is needed.

3. Making Tangible Images

Augmenting images with haptic models requires for answering two questions: where to obtain the corresponding models and how to match them with the respective parts of the images.

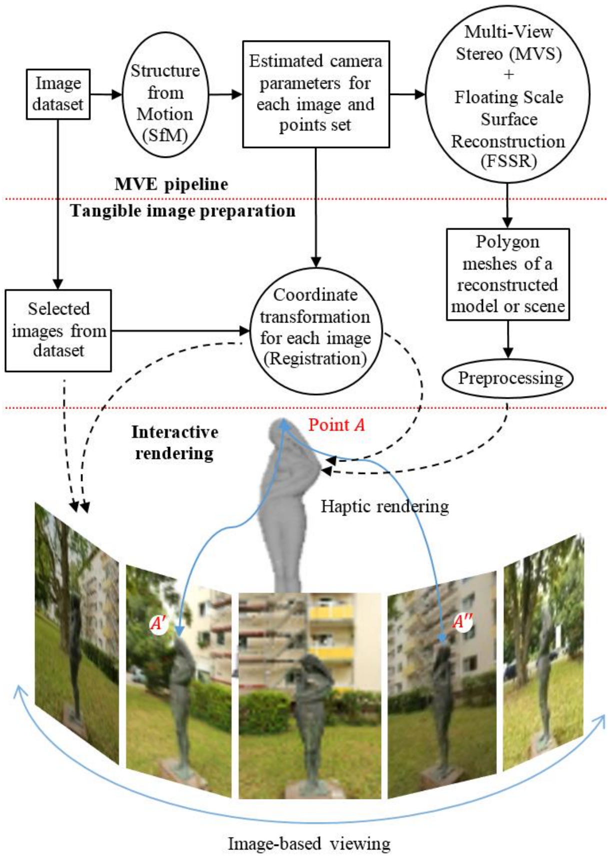


Fig. 2. In the MVE pipeline, Structure-from-Motion (SfM) techniques are used to reconstruct camera parameters and a sparse points set. Then a mesh is reconstructed using Multi-View Stereo (MVS) and Floating Scale Surface Reconstruction (FSSR) approach. In tangible image pipeline, the reconstructed model is matched with corresponding images to provide haptic feedback. Rotation of the scene can be simulated by a series of selected images.

There are several ways to obtain models of a real scene, such as interactive modeling of the scene in computer-aided design systems, reproducing the model based on the data collected from 3D scanners and reconstructing the model based on multi-view reconstruction methods. Matching a model with an image requires for taking into account its perspective distortions: we may either define the model in a perspectivally distorted modeling space matching the image coordinate space as in [2], or use camera projection transformation for mapping coordinates between the image and the model coordinate spaces.

In this paper, we use reconstructed models to make the corresponding image dataset tangible. Models generated from MVE [9] are used as examples. Given multiple images of a real scene, MVE reconstructs a polygon mesh of the scene, along with the estimated camera parameters for each input image (as shown in the MVE pipeline in Fig. 2). If the input image dataset contains close-up photos, the output can be a high-resolution 3D scene with millions of polygons and some regions are of a higher resolution than other parts.

In the pipeline of tangible image (as illustrated in Fig. 2), we simulate virtual walkthroughs in the real scene with a series of selected images from the dataset. Then the reconstructed model is registered with each image using the estimated camera parameters and the respective coordinate transformation. To incorporate reconstructed meshes in haptic interaction, the worst case scenario is considered in this paper, i.e. we show an approach to haptically rendering large-scale imperfect meshes. This approach is pluggable and can be used for haptic rendering with any large-scale meshes. It performs the following tasks:

- **Coordinate transformation**. We register the haptic display with the photo using the reconstructed camera parameters.
- **Preprocessing**. We deal with the imperfections of the reconstructed mesh and build acceleration structures for collision detection.
- **Haptic rendering**. We propose a hybrid collision detection algorithm to handle collision detection with large-scale meshes and explain how to render force feedback based on the collision results.

3.1. Coordinate transformation

When using images to replace visual rendering of the meshes, we need to match the haptic models with the images so that the image content matches the haptic display. In a multi-view reconstruction process, camera parameters of the images can be estimated based on structure-from-motion techniques [17]. Therefore, given a target image and corresponding reconstructed model, the estimated camera parameters could be used to calculate the *modelview* and *projection* matrices for projecting the model in the camera frustum. Suppose R_C is the orientation matrix of the virtual camera with respect to the world coordinate system, T_C is the column vector which defines the location of the virtual camera in the world coordinate system, f is the focal length of the camera, *img_width* and *img_height* are the width and the height of the given image, pp_x and pp_y are x, y coordinates of the principal point offset of the camera in pixel coordinate system, z_{near} and z_{far} are the z coordinates of the near and far clipping planes, then the 4*4 modelview and projection matrices M_{mol} and M_{proj} can be obtained as follows:

$$M_{mol} = \begin{pmatrix} R_C & T_C \\ 0 & 1 \end{pmatrix} \quad (2)$$

$$M_{proj} = \begin{pmatrix} 2f\alpha_x & 0 & 2(pp_x - 0.5) & 0 \\ 0 & 2f\alpha_y & 2(pp_y - 0.5) & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{far}-z_{near}} & \frac{-2\, z_{far}\, z_{near}}{z_{far}-z_{near}} \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (3)$$

$$aspect = img\_width / img\_height \quad (4)$$

$$\alpha_x = \begin{cases} 1, & if\ aspect > 1 \\ 1/aspect, & if\ aspect \le 1 \end{cases} \quad (5)$$

$$\alpha_y = \begin{cases} aspect, & if\ aspect > 1 \\ 1, & if\ aspect \le 1 \end{cases} \quad (6)$$

Note that the origin of the image coordinate system for the MVE-produced models is at the top-left corner of the image while it is at the bottom-left corner of the image in OpenGL. Therefore when displaying MVE models in OpenGL, the y-axis needs to be inverted to match the image. This could be done by inverting all elements in the second row of either M_{mol} or M_{proj}.

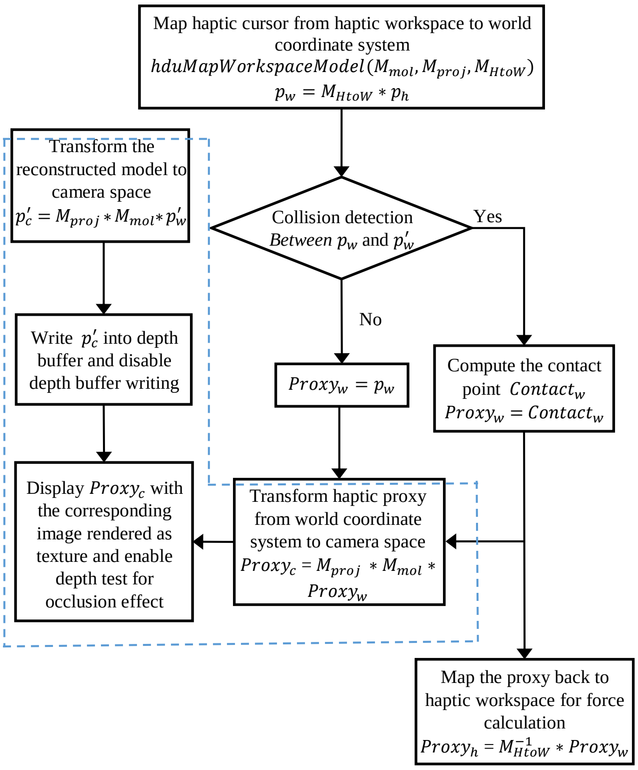


Fig. 3. Flowchart of the mapping process.

There are three workspaces involved in the visual-haptic interaction: the camera workspace (defined during the structure-from-motion process), the haptic workspace, and the world coordinate system. The whole mapping and transformation process behind the interaction scene is illustrated in the flowchart in Fig. 3. The procedures enclosed by the blue dashed lines are for visual rendering. In real 3D scenes, the haptic cursor would be hidden when moving to the back of the objects. To simulate such occlusion effect with displaying only 2D images, we write the reconstructed models to the depth buffer and then disable writing to the depth buffer right after the writing operation. The depth buffer writing is kept disabled in the following rendering loop. Afterwards, with depth test enabled and *glDepthFunc* depth comparison function set to GL_LEQUAL, the depth values of the models rendered in real-time (e.g., haptic cursor) are compared with the depth values stored in the depth buffer. A pixel of the haptic cursor is only drawn if the incoming depth value at this pixel is less than or equal to the stored depth value. In such a way, if the haptic cursor goes to the back of the reconstructed model (i.e. the incoming depth value is greater than the stored depth value), it is not drawn and the occlusion effect is thus achieved.

In the haptic servo loop thread, the position of the haptic cursor is mapped to the world coordinate system for collision detection and then mapped back to the haptic workspace for force rendering if the collision happens. The generated proxy position is transformed to the camera workspace and sent to the client thread for displaying.

3.2. Preprocessing of the reconstructed mesh

In order to apply the collision detection algorithm, we need to preprocess the reconstructed mesh, which includes three steps.

The first step is to handle imperfections with regard to duplicate vertices inside the mesh. Reconstructed models are likely to have duplicate vertices, e.g., the city wall model in Fig. 9(a) has 1883 groups of duplicate vertices. These vertices cause the appearance of holes during haptic rendering leading to pop-throughs during the haptic interaction. We therefore delete the duplicate vertices and zero-area polygons in the mesh in the following way. All the vertices are traversed to form a list of duplicate vertex groups, and in each group the vertex with the smallest index is considered as effective while the others are deemed duplicates. Then, the polygons with duplicate vertices are divided into two groups. Those with two or more duplicate vertices from the same group (i.e. zero-area facets) are deleted directly, while the others have their duplicate vertices replaced by the effective vertices of the same group.

After removing all the duplicate vertices and zero-area polygons, the second step is to build the connectivities among vertices, line segments and polygons and store all the neighbors for each primitive. With reference to the "neighborhood watch" algorithm [16], there are three kinds of primitives in a mesh: vertices, line segments and polygons. Thus the concept of *active polygon* is extended to *active primitive*, the primitive that the HIP is in contact with. In our paper, we define the *neighbors* for the three primitive types referring to the definitions in [16]:

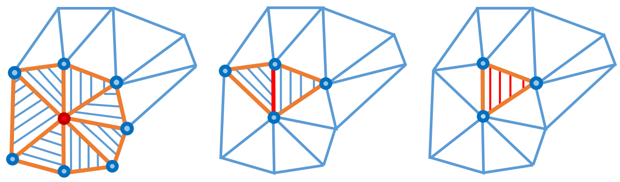
- For a polygon, the neighbors are its line and vertex components.
- For a vertex and a line, their neighbors include all the polygons connected to it and all the lines and vertices that comprise these polygons.

Fig. 4 illustrates an example of how neighbors are defined for a vertex, a line segment and a polygon.

Based on the connectivities between the vertices and polygons, the vertex normals are recalculated by summing up the weighted normal of the neighboring polygons and normalizing the sum [18] as in (1).

$$n_v = \frac{\sum_i \alpha_i * n_{f,i}}{\left\| \sum_i \alpha_i * n_{f,i} \right\|} \qquad (1)$$

Here, the weight is each neighboring polygon's inner angle at this vertex. Besides, we also check and store whether a line is on a convex or concave surface. The lines with only one adjacent polygon are marked as edges. These lines may be the edges of the outer contour or the edges of holes on the surface of the mesh.



(a) (b) (c)

Fig. 4 . A vertex neighbor is marked as a small circle, a line segment neighbor is marked in orange color and a polygon neighbor is marked with stripes. (a) The red vertex has 7 polygon neighbors, 14 line segment neighbors and 7 vertex neighbors. (b) The red line segment has 2 polygon neighbors, 4 line segment neighbors and 4 vertex neighbors. (c) The red polygon has 3 line segment neighbors and 3 vertex neighbors.

In the final preprocessing step, we apply a uniform partition to the space within the bounding box of the polygon mesh and divide this space into cells. The size of the cell is determined by the highest local density of the mesh. To narrow down the search range for active primitive and to meet the real-time requirement, the maximum number of polygons in one cell needs to be constrained. We identify the largest number of polygons in one cell before proceeding to collision detection and adjust the cell size based on this number. In our method, a polygon is considered as belonging to one cell if a vertex of the polygon is in this cell, the polygon has an edge intersecting with the bounding box of this cell or the bounding box of this cell intersects with the polygon. This criterion is the same as that in [11].

3.3. Collision detection with the preprocessed meshes

The challenge of collision detection with large-scale meshes lies in how to obtain the active polygon in real time (1000 Hz). The existence of an active primitive is the necessary and sufficient condition for point-based collision. As illustrated in Fig. 5, in our method the detection procedure in the current frame is divided into two branches based on the collision status in the immediately preceding frame.

If there is no collision between the HIP and the mesh in the previous frame (*the first branch*), we check whether the ray from the HIP in the previous frame to that in current frame intersects with the mesh. The reason behind it is that when the HIP goes inside of the mesh from outside, intersection always happens. Therefore, based on the intersection test result, we further break down this branch into two sub-branches:

1. If the ray from the previous HIP to the current HIP intersects with the mesh at one polygon, this polygon is treated like the previous active primitive and served as start point in the tracking for active primitive in the current frame.

2. Naturally, if there is no intersection then there is no collision in the current frame.

Correspondingly, if the HIP collides with the mesh in the previous frame (*the second branch*), then the active primitive in the previous frame is used as a start point to track the path of the HIP and locate the active primitive in the current frame. If the tracking succeeds, it means that the HIP is still in contact with the mesh in this frame. Otherwise we consider that the contact has stopped.

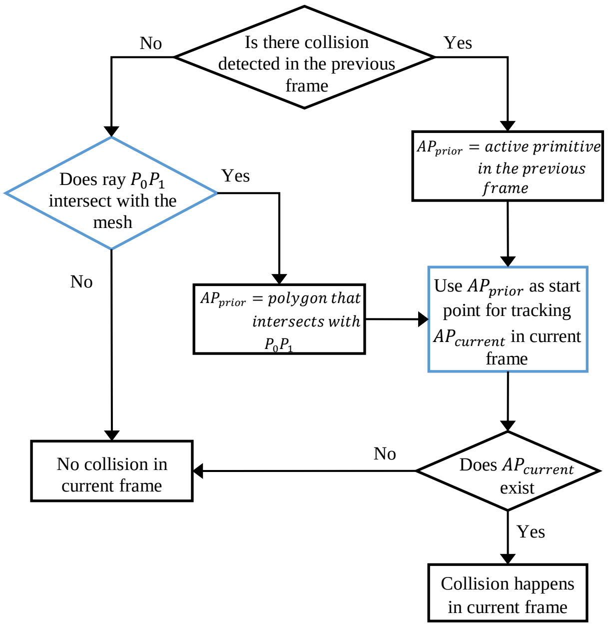


Fig. 5. Flowchart of collision detection process. P_0, P_1 denotes the HIP in the previous and the current frame. *AP* means active primitive.

During the whole process, there are two key modules: the intersection test between the ray and the mesh and the tracking of the active primitive (marked blue in Fig. 5). More implementation details about these two modules are presented in the following.

3.3.1. Intersection test

In our previous paper [3], the collision detection algorithm is built on the assumption that if the HIP crosses mesh surface in a frame then the active polygon in this frame would be in the same cell as the haptic cursor. This assumption enables us to narrow down the detection range, however, it does not always hold. When it fails, the detection would also fail, resulting in unexpected pop-throughs.

To remove this assumption, in this paper we introduce ray tracing into the first branch of our algorithm, dismantling this part into an intersection test, which will be described in the following, and a tracking process, which is the same as the process run in the second branch but with different initial values.

For the intersection test, the first step is to check whether the HIP is inside the bounding box of the mesh in the current frame. If it is inside the bounding box, we proceed to locate the cell that the HIP is in. Suppose P_0 is the HIP in the previous frame and P_1 is the HIP in current frame. If $P_1 \in cell_0$, then based on the connectivity relation between cells we can find all the cells $\{cell_0, \ldots, cell_n\}$ that the ray P_0P_1 passes through. To find the intersected polygon from these cells, we start with $cell_0$. We check whether ray P_0P_1 intersects with any of the polygons inside $cell_0$. If this is the case, we check whether there is an intersection with polygons inside $cell_1$. We continue like this until we find the intersected polygon or we reach $cell_n$. In this way, the computation complexity of the intersection test is only related to the polygon number inside the cells along the HIP path.

Fig.6 illustrates how we derive all the target cells one by one. As we can see, P_1 is in cell a and the ray P_0P_1 intersects with the blue polygon at point Q. This intersected polygon is in cell b, d and e, not in the same cell as the HIP P_1. Since cell a does not contain the intersected polygon, we check whether P_0P_1 intersects with the boundary of cell a. Since an intersection exists, we locate the intersection point P_1^1 and update P_1 with it. The location of this intersection point also determines the common face and thus the next target cell c. In the same manner, we can identify cell b based on intersection point P_1^2 and eventually obtain all the cells $\{a, c, b, e, d\}$ in the listed order.

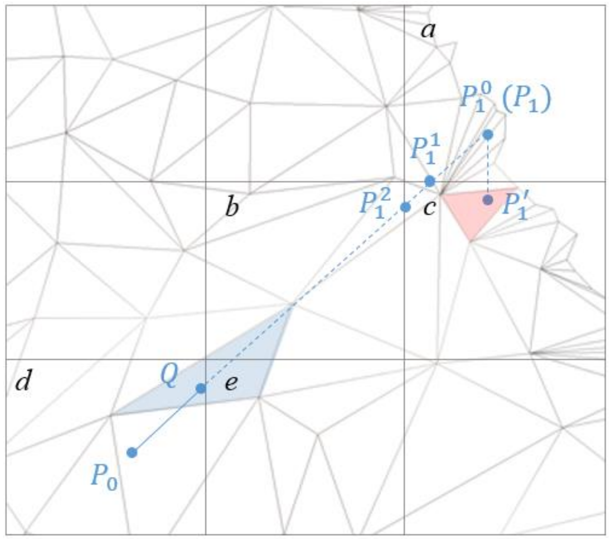


Fig. 6. An example to illustrate how to find all the cells intersected with ray P_0P_1 in the following order: $a \to c \to b \to e \to d$. The triangle intersected with P_0P_1 is marked blue while the active primitive is marked red. P_1' is the projection of P_1 on the active primitive.

We note that the existence of an intersected polygon does not necessarily mean there is collision between the HIP and the mesh in this frame. Let us consider as an example the case in Fig. 7. The ray P_0P_1 intersects with the mesh, but neither P_0 nor P_1 is inside the mesh, i.e. no collision happens. Therefore, after we obtain the intersected polygon, we need to use it as start point to track the active primitive. Only if an active primitive exists can we confirm that the collision has happened.

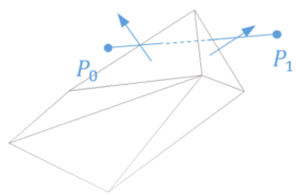


Fig. 7. An example to illustrate difference between intersection and collision.

3.3.2. Tracking of the active primitive

Based on the geometry connectivities built in preprocessing step, given a start point, we are able to follow the path of the HIP and track the active primitive. This start point can be a polygon, a line segment or a vertex. We refer to it as a *start primitive* in the following. The start primitive can be obtained from two sources: the intersected polygon derived from the intersection test or the active primitive in the previous frame.

Three conditions need to be fulfilled to make a primitive active in one frame:

- **HIP criterion**: the HIP is inside the mesh.
- **Distance criterion**: this primitive has the shortest distance to the HIP compared to its neighbors.
- **Projection criterion**: the orthogonal projection of the HIP onto this primitive is inside its range.

Considering the relations between these three conditions, we examine them in the following order: firstly, we find the primitive which meets the last two conditions, then we check whether the first condition is true for this primitive.

```
Algorithm 1 Algorithm for obtaining the active primitive
activepri_prior ← startprimitive

repeat
  activepri_temp ← activepri_prior
  set A = {activepri_prior, neighbors of activepri_prior}

  for all polygons ∈ A,
  if there exists a polygon which has the projection of HIP onto it inside its range then
    distmin = min{|dist_i| :the projection of HIP on polygon_i plane is inside polygon_i, polygon_i ∈ A}
    activepri_temp ← polygon with distmin to the HIP
  else
    for all line segments and vertices ∈ A,
    if min{dist_i : line_i ∈ A} < min{dist_j : vertex_j ∈ A} then
      distmin = min{dist_i : line_i ∈ A}
      activepri_temp ← line segment with distmin to the HIP
    else
      distmin = min{dist_j : vertex_j ∈ A}
      activepri_temp ← vertex with distmin to the HIP
    end if
  end if

until activepri_temp == activepri_prior

vector ← vector from HIP to the projection of HIP onto activepri_temp
normal ← the normal of activepri_temp
if vector * normal < 0 then
  collision ← FALSE
else
  collision ← TRUE
  activeprimitive ← activepri_temp
end if
```

Fig. 8. Psedocode of algorithm for obtaining the active primitive.

The whole tracking process is represented as a repeat until loop operation in the pseudocode given in Fig. 8. The loop starts with the determined start primitive. In each iteration, AP_{new} is selected from the input primitive AP_{prior} and its neighbors based on the distance and projection criteria for being an active primitive. If AP_{new} is the same as AP_{prior}, it would be considered as a potential active primitive and be checked to find whether it meets the last condition, i.e. the HIP criterion. Otherwise, the loop continues with AP_{new} as the input primitive for the next iteration. A primitive that meets all three criteria is the active primitive in the current frame and it will be saved and used as the start primitive for the tracking in the next frame.

In our algorithm, when examining a primitive and its neighbors based on the distance and projection criteria, we incorporate the features of each primitive type into the checking order. For a polygon, if the projection of the HIP is inside its range, then it definitely has the shortest distance to the HIP compared to its components (three line segments and three vertices). The same rule applies to the line segment: If one line segment has the projection of the HIP on it, it certainly has the shortest distance to the HIP compared to its two vertices. Therefore, we calculate and compare the distances of the potential active primitive and its neighbors to the HIP following this order: polygons first, then line segments, and lastly the vertices (reflected in the blue part of Fig. 8).

3.4. Force rendering

We assume that the interactive models are hard and stiff objects, therefore we apply constraint-based haptic rendering: we compute a proxy to represent the haptic cursor so that the cursor is always visible. When the HIP is moving in free space, the position of the proxy matches the HIP. When there is a collision, the active primitive is known and the proxy is assigned as the projection of the HIP on the active primitive.

We use a spring force model. The magnitude of the force feedback is proportional to the penetration depth of the HIP into the active primitive, which is exactly the *distmin* that we obtain in the iteration loop of Algorithm 1. Normally, the force is computed in the same direction as the facet normal. In our method, we use this approach if the active primitive is a polygon. When the active primitive is a line segment, the force is applied along the direction opposite to the movement, which is from the proxy to the HIP position. In this way, we can effectively prevent the haptic cursor from crossing the edges. Thus, if the cursor slides to a hole on the mesh, it would not fall into the hole. The disadvantage of this strategy is that if the cursor slides along a ragged edge, there are frequent changes in the force direction, since we always give the cursor a resistant force perpendicular to the edge. If the force direction is in the same direction as the velocity, this may lead to a cursor jump.

4. Results

The images in Fig. 9 illustrate how the concepts introduced in the previous section are implemented given a reconstructed model. Fig. 9(a) shows the original reconstructed city-wall model included in MVE [9], while the small image in the left upper corner is the image to be used for visual display in the interactive scene. Based on the reconstructed camera parameters of this image, we transform the model to the camera workspace and obtain the part in Fig. 9(b) after clipping. We can see that the clipped model matches with the content of the image (Fig. 9(c)). After transformation and mapping, the haptic cursor is able to interact with the city wall in the image as displayed in Fig. 9(d). The red ball in Fig. 9(d) represents the proxy of the haptic cursor. A red line pointing to the normal direction is also shown, indicating that the cursor is in contact with the model now.

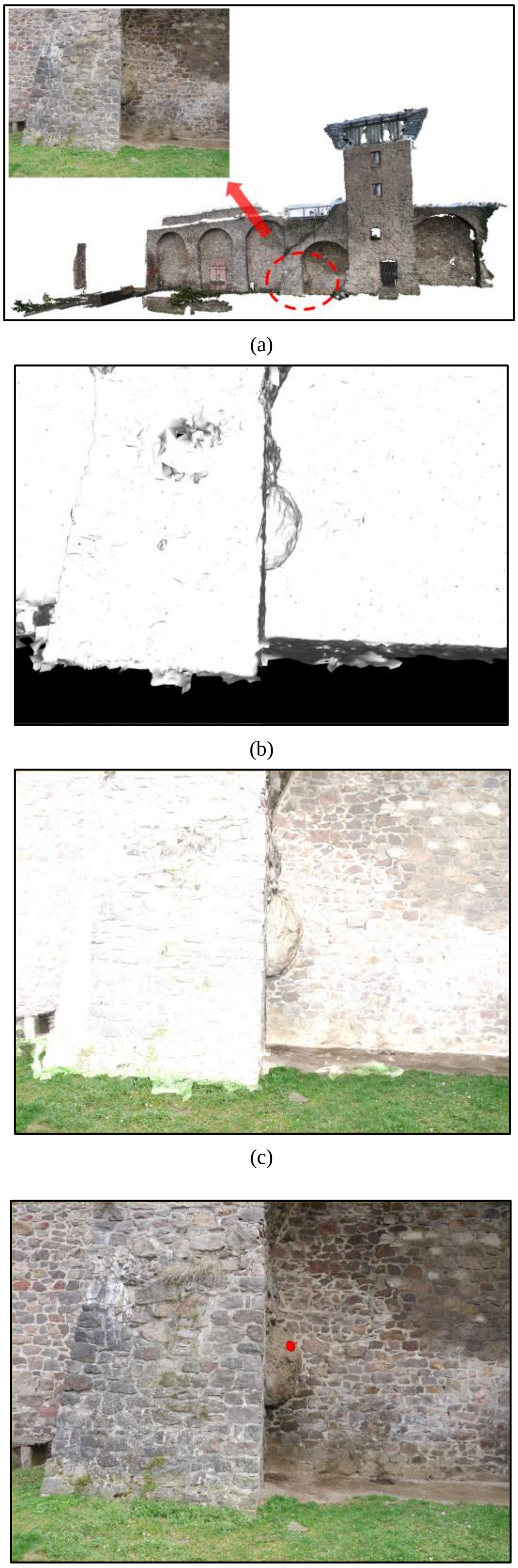


Fig. 9. (a) the original reconstructed model. (b) the transformed model displayed in simulated camera frustum. (c) the alignment of the transformed model and the image. (d) a snapshot of the interactive scene.

The examples of haptic interaction with the models reconstructed from images (Fig. 10) can be seen in the companion video, which is also available at https://youtu.be/6_tHrG9q3H8. We are able to explore the scene by switching between consecutive images forming a walkthrough and touching the image content with the haptic cursor. With the reconstructed mesh superimposed on the images, the images are tangible like real 3D scenes. When the haptic cursor collides with a tangible object in the image, it always stays on the surface of the object as if it is interacting with real rigid objects. When the cursor goes to the back of the object, it would be hidden.

Fig. 10. Examples of interactions with the models reconstructed from images. The cursor is displayed as a red ball in the interactive scenes.

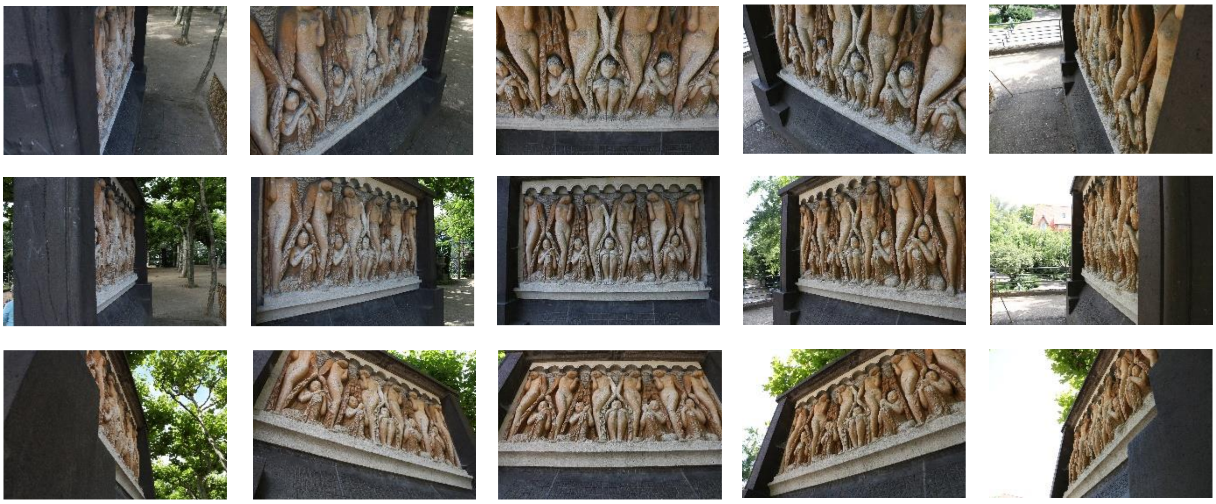


Fig. 11. Top row: photos taken from high viewpoint. Middle row: photos taken from normal eye-level viewpoint. Bottom row: photos taken from low viewpoint.

5. User Study

In our previous paper [3] we conducted the comparison experiment which has shown that the performance of our system far outweighs the commonly-used haptic renderers (God-object renderer [10] provided by H3D API and OpenHaptics HLAPI [13]) in colliding with large-scale meshes. In this paper we report the results of the subjective user tests evaluating what the users think about our approach.

5.1. Capturing test photos

Mathildenhöhe sculpture photos (Fig. 11) used in this test were captured by orbiting a camera around the sculpture center. The camera was incrementally rotated to record the sculpture from different viewing angles. Besides taking photos from normal eye-level viewpoint, we also captured the sculpture from high and low viewpoints. During capturing the camera was always looking at the central part of the sculpture.

Selectively we chose 21 photos from each viewpoint and put them in a 3-row grid to simulate a constrained rotation effect (Fig. 11). All these chosen photos were preloaded to our system before the test.

In the reconstruction of the Mathildenhöhe sculpture model, 256 photos were put into the MVE system, including the photos used in our test. The reconstructed model contained around 5 million triangles.

5.2. Experimental Setup

Our system was run on a computer with CPU working at 2.60GHz. The users were expected to learn the displayed scene by both visual and haptic interaction with it. The visual interaction was supported as a panoramic rotation of the scene controlled by the left and right arrow keys. With each key pressed, the respective next image of the captured scene from the image sequence was displayed. Haptic interaction was implemented using Geomagic Touch desktop haptic device placed close to the user's dominant hand (Fig. 12). The users sat in front of the device and were asked to touch the objects in the scene by moving the haptic cursor displayed in it. The scene could be rotated in 180 degrees counterclockwise to view and touch the objects from different perspective.

Fig. 12. A beta test participant interacts with the tangible photos.

5.3. Experimental Design

5.3.1. Measurements

Table 1

Questions and corresponding factors. These factors are rated on a scale of 1 to 5, where 1 means not at all and 5 means very much.

Question	Factor
How realistic is your haptic interaction with the displayed scenes?	Realism
How well could you actively explore the displayed scenes by touching?	Realism, Sensory
How comfortable do you feel interacting with the displayed scenes?	Comfort
How useful is the haptic feedback in improving your interaction experience?	Sensory
How satisfied are you with your interaction experience?	Satisfaction

A questionnaire as in Table 1 was designed to evaluate interaction with tangible images. Based on Presence [19], four Factors are evaluated in this questionnaire: realism, sensory, comfort and satisfaction.

Among the five questions, the second question contributes to two factors. According to [19], the correlation coefficient of this question is 0.15. Thus we computed the results for realism and sensory in this way:

$$realism = 0.15 * Q2 + 0.85 * Q1 \quad (7)$$

$$sensory = 0.15 * Q2 + 0.85 * Q4 \quad (8)$$

5.3.2. Procedures

24 users participated in our test, 7 female and 17 male. 1 participant was ambidextrous and tried our system with both hands. 17 of them never used any haptic device. The entire test took 20 to 30 minutes to complete. Here are detailed procedures:

1. Demonstration of how to use Geomagic Touch with an example. Proper training is necessary before the test to eliminate the tension of the users, especially for novices.
2. User testing. The users were asked to explore the displayed image scene with the haptic device. Viewpoints and viewing angle can be changed by pressing arrow keys.
3. Filling in the questionnaire.
4. Collection of oral feedback. This step is for gaining a more comprehensive understanding of the ratings. Their answers are recorded on the questionnaire during the collection.

5.4. Results

The results of the questionnaire are shown in Table 2. The goal of this user test is to know what users think of our system, and more specifically, to assess the likelihood that users would accept and want to use our system. We can see from the table that the means for the four factors were all above 3 (neutral), which reflects a positive attitude towards the system. If we calculate the true population means, the results are still positive. Let us consider realism, the factor with the lowest mean, as an example. The 95% confidence interval for its mean 3.46 is 3.05 to 3.87, of which the lower bound is still slightly higher than 3 (neutral).

Table 2

Results of the questionnaire. These factors are rated on a scale of 1 to 5, where 1 means not at all and 5 means very much.

Factor	Mean	Standard deviation	95% Confidence Interval
Realism	3.46	0.98	0.41
Comfort	3.58	1.14	0.48
Sensory	3.64	0.90	0.38
Satisfaction	3.58	1.02	0.43

5.5. Discussion

Comments from users are categorized into four groups.

5.5.1. Pleasure

Most users found it impressive to feel the depth of the object in the photo, especially when experiencing significant changes in depth, e.g., sliding from a platform away from us to one closer to us (as in Fig. 13). Besides, we got comments that they enjoyed this user test and would like to try our system again.

Fig. 13. Example of sliding from far surface to near surface. The haptic cursor trace is marked with cursor sample points (sampled at 20 Hz), which are represented as red balls. The red line always points to the force direction.

5.5.2. Force feedback

Most users encountered problems while sliding the cursor on the surface of small structures with large curvature, because they found it hard to constrain the cursor to the surface. Two of them suggested that we should provide a zooming operation so that they could touch small details better. Another user compared this phenomenon to the real life situations and explained it as lack of automatic assistant force from the wrist which we obtain when sliding our figure on a real curve.

Six users expected to feel the physical properties of the objects in the interaction, e.g., stiffness, friction, texture and viscosity. Constrained by the device, it is impossible to simulate interaction with rigid bodies, but in the future we could make force feedback more realistic by adding haptic texture and viscosity to the models and applying friction based on the real material properties.

Another interesting finding from the users' feedback is that most of them believe that there is too much roughness at some places which are supposed to be smooth. This may reflect an unconscious relation between visual feedback and haptic feedback. The users have an expectation about what the haptic feedback should be like based on what they see in the photos. If they do not visually perceive the details that they are touching, they are likely to deny these details and interpret them as unexpected roughness. This partially explains why the average rating on realism is just mediocre (3.46 on a scale of 1 to 5). Based on this, we conclude that such a system should not provide haptic details that cannot be perceived by eyes. In addition, the force should be smoothed so that the users do not get frustrated because the HIP is stuck at small surface details.

5.5.3. Device

Five users pointed out that they felt tired or uncomfortable holding the handle for a prolonged time and three of them explicitly wrote that this has negative influence on their ratings for satisfaction. We could not change the ergonomics of the device but there could be some ways to improve the comfort level, e.g., using some form of cushioning or support for the hand.

Another complaint about the device is that it is not so intuitive, which results from limited force output and only one interaction point. These are limitations of such ground-based haptic interfaces. If we replace the device with body-based haptic interfaces such as gloves, suits and exoskeletal devices, the user experience could be improved to some extent, but the cost would also increase largely.

5.5.4. Usefulness

Most users showed reserved positive attitude towards the usefulness of the haptic feedback in interaction with photos. Only three out of twenty-four users gave negative feedback.

Those who gave positive or neutral feedback believed that having one more dimension of feedback is better than simply viewing the photos. They commented that this system could be useful for people with bad depth perception or if the photo content involves unclear structure. One user also mentioned an inspiring observation: her memory about photos is largely enriched in this way and she can remember the content of the photos better after touching them.

5.5.5. Others

Before the test, we did not inform the users which part of the photos is tangible, so they need to explore it themselves. Three users found that only the sculpture part is tangible and commented that they also wanted to touch other objects in the photo background, e.g., trees, houses and cars. Therefore, one of our goals in the future is to make the whole photo tangible or to think of a way to communicate to the users which parts are tangible.

Moreover, we noticed that two users were confused about what touching feels like at the beginning of the test. After our explanation they knew that seeing the haptic cursor does not indicate the occurrence of contact with the objects in the scene. They would feel the haptic feedback only when reaching the depth of the object with the cursor. This confusion is due to the fact that people are not used to derive depth information in the virtual environment without reference. Therefore, additional training about what it means to touch might be necessary and assistive visual feedback could be helpful.

6. Conclusion

We have presented our approach to creating tangible images using models reconstructed by multi-view vision techniques. To deal with large-size, partially dense reconstructed meshes, we propose an improved hybrid collision detection method. By preprocessing the mesh with uniform partitioning and building connectivities among the vertices, lines and polygons, we are able to handle collision detection with meshes of over ten million triangles.

In this approach, we align the haptic models with the images so that the haptic display would match the visual content. Occlusion of the haptic cursor is simulated as if it was interacting with a real 3D scene.

With the presented method, we add a new modality into interaction with images. Besides viewing an image, this method enables us to appreciate the image content within a touching distance and complements our viewing experience.

Despite the limitations of the device (i.e. not so intuitive, feeling uncomfortable if holding the handle for long time), the results of the usability test show that we have provided an enjoyable and easy way to enrich images with a touch interface and haptic feedback. Based on the users' comments, there are many things that can be improved (e.g., adding haptic texture and viscosity to the models), but generally this new approach meets the users' expectation about haptic interaction and it brings new possibilities into interaction with images.

ACKNOWLEDGMENT

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centers in Singapore Funding Initiative, joint PhD Degree Program NTU-TU Darmstadt, and MOE Singapore Funding RG17/15 "Haptic Interaction with Images and Videos". The authors also thank Mr. Patrick Seemann and Mr. Stepan Konrad for providing the photos and the models used in the user test.

REFERENCES

[1] Rasool S, Sourin A. Image-driven Haptic Rendering. In: Transactions on Computational Science XXIII, Journal Subline LNCS 8490; 2014. p. 58-77.

[2] Zhang XZ, Sourin A. Image-inspired haptic interaction. Computer Animation and Virtual Worlds; 2015, 26: p. 311-319.

[3] Zhang XZ, Goesele M, Sourin A. Haptic interaction with a polygon mesh reconstructed from images. In: Cyberworlds(CW) 2016; p. 49-56.

[4] OTOY. Brigade. 2016. Available: https://home.otoy.com/render/brigade/.

[5] Morris D, Joshi N. Hybrid rendering for interactive virtual scenes. In: Stanford University Technical Report CSTR; 2006, 6.

[6] Otaduy MA, Jain N, Sud A, Lin MC. Haptic display of interaction between textured models. In: Visualization IEEE; 2004. p. 297-304.

[7] Rasool S, Sourin A. Tangible images. In: SIGGRAPH Asia 2011 Sketches; p. 41.

[8] Kim SC, Kyung KU, Kwon DS. Haptic annotation for an interactive image. In: Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, ACM; 2011.

[9] Fuhrmann S, Langguth F, Moehrle N, Waechter M, Goesele M. MVE – An image-based reconstruction environment. Computers & Graphics; 2015, 53: p. 44-53.

[10] Zilles CB, Salisbury JK. A constraint-based god-object method for haptic display. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems; 1995, p. 146-151.

[11] Ruspini DC, Kolarov K, Khatib O. The haptic display of complex graphical environments. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques; 1997, p. 345-352.

[12] *CHAI 3D*. 2014. Available: http://www.chai3d.org/index.html

[13] Geomagic. 2016. *OpenHaptic Toolkit Overview*. Available: http://www.geomagic.com/en/products/open-haptics/overview

[14] Gregory A, Lin MC, Gottschalk S, Taylor R. H-collide: a framework for fast and accurate collision detection for haptic interaction. In: proceedings of Virtual Reality conference 1999; 1999.

[15] Anderson T, Brown N. The activepolygon polygonal algorithm for haptic force generation. In: Proceedings of the sixth PHANToM Users Group Workshop, 2001.

[16] Ho CH, Basdogan C, Srinivasan MA. Efficient point-based rendering techniques for haptic display of virtual objects. Presence; 1999, 8(5): p. 477-491.

[17] Szeliski R. Computer vision: algorithms and applications. Springer Science & Business Media; 2010.

[18] Thürrner G, Wüthrich CA. Computing vertex normals from polygonal facets. Journal of Graphics Tools; 1998, 3(1): p.43-46.

[19] Witmer BG, Singer MJ. Measuring presence in virtual environments: A presence questionnaire. Presence; 1998, p. 225–240.