Haptical Rendering of Triangle Meshes with Fixed Direction Hulls

Zdeněk Kabeláč

Human-Computer Interaction Laboratory
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
kabi@fi.muni.cz

Abstract

We present the system for the force–feedback visualization, which uses hard realtime operating system. Software architecture uses two cooperating tasks. The first one autonomosly resolves forces for haptic contact within the small set of triangles and it maintains stable high update rate in the range of several kHz. The second task runs in order of magnitude slower. It selects triangle sets contained in limited subspace from the virtual scene with the help of Fixed Direction Hulls.

Keyword: PHANTomM, haptics, RT-Linux, virtual environment, fixed direction hull, force feedback

1 Introduction

To get the sense of touch we add haptical rendering to the graphical visualization. For haptical visualization we use force–feedback device PHANTomM. For the best experience graphical and haptical systems have to run fast and fluently. Processing complicated analytic surfaces like NURBS[1] is still highly computation intensive task. As we use OpenGL for graphical visualization it is natural to use uniform representation based on triangle sets and meshes. Hence we triangulate all complex models including analytic surfaces before we visualize them.

In this article we present our approach for haptical visualization of large triangle sets on PHANTomM device using hard realtime operating system RT–Linux[1].

With this system we try to avoid the computation of complicated surfaces in the realtime as they could be precomputed into triangle meshes without the notable loss of quality.

We have decided when we have already triangulated objects for 3D rendering, it would be good to use the similar representation (preferably the same one) also for haptical rendering. Simple geometrical objects like spheres, toroids and various other shapes, which could be described by algebraic equations, are more precisely haptically rendered using the analytic approach. However in the real life scenes it is hard to find such objects and we usually work with huge triangle meshes from 3D modelling tools. Moreover these triangle sets commonly contain problematic areas and its not even specified what it means to be inside or outside of an object.

Some researchers have already presented their algorithms for haptical rendering of triangles, but these are usually not prepared to work reliable with update frequencies in the range of a few kHz.

In this article we also mention some preprocessing steps we use to minimalize problems with more or less incorrect triangle models. These are usually being generated by various 3D modelling tools.

[1] http://www.rtlinux.org

2 Previous Work

To achieve hard surface with haptical rendering we need high update rates (at least 1kHz) and fast methods for evaluation of reacting forces. Contact points necessary to compute forces are usually found with the help of collision detection systems. That means we have to use very efficient and fast algorithms here.

A good overview of traditionally used methods including oriented bounded boxes (OBB) and axis aligned bounding boxes (AABB) and bounding volume hierarchies (BVH) can be found in [2]. For the haptical rendering few special collision detection systems have been developed: H–Collide, I–Collide see: [3]. These systems are usually based on OBB and they focus on detection of one surface contact point.

For the system with two levels of computation hierarchy similar approach could be found in ArmLib[4]. This project uses an equation of a single plane which is maintained by the fast internal loop while the second task updates this plane with much slower rate.

3 Method Overview

Our goal was to develop the method which could be used generally to haptically visualize the same data sets as are used for graphical visualization.

The algorithm we present here, is divided into two parts running in two different environments. The first part is running in the normal user space mode and its purpose is to select a small subset of triangles from the large virtual scene. The second part is running in the realtime environment and has limited capabilities. The computation time is bounded and also the memory access is limited in this environment. So the algorithm has to be simple and fast. This task works with some small neighbourhood for a given point and surrounding triangles proportionally influence the resulting force.

Using two separate algorithms where one is used to pick small surrounding virtual environment from the large one and the second task is resolving haptic feedback within this smaller subset, might be viewed as a special case of 3D traversal exploiting frame-to-frame coherence. For the hierarchical representation of the scene we use BVH of FDH which will be briefly described later. We use two levels of BVHs. The first level contains all objects and it is used to detect collision between them. The second one is used to select few triangles from the colliding object.

4 Algorithm

In this section we describe how the system processes triangle sets. We present new algorithm used for calculation of reacting forces which also does smooth shading of normal vectors.

4.1 Preprocessing

For OpenGL rendering number of various optimalization have been developed. With todays graphics cards we do not have to care too much if we will draw several hundred triangles more or less. However, with the haptic rendering the situation is different. We need to eliminate as much triangles as we can before processing them in the force loop.

Usually it is better when the scene consists of large triangles. But at certain size large triangles cause the problem as they need too large bounding volumes. To create more compact bounding volumes we have to split larger triangles into the smaller ones. Small triangles can be more easily sorted in a balanced BVH.

On the other hand too small triangles which might appear in some detailed models have to be reduced otherwise the selected environment would be too small and user would shortly escape from 'precached' triangle set. For this purpose right now we preprocess triangle sets with surface simplification algorithm[5].

Preprocessing algorithm consists of following steps:

- create the list of triangles and find common edges
- with respect to sharp edges fix or fill missing vertex normals
- check if adjanced triangles have the same normal vectors in common vertices; in the case normals differ add two fake triangles (fig.1) with only two different points which will assure continuous change of the normal vector.

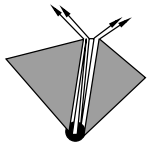

Figure 1: Adding two fake empty triangles.

- split large triangles
- final validation of normal vectors

4.2 Computation

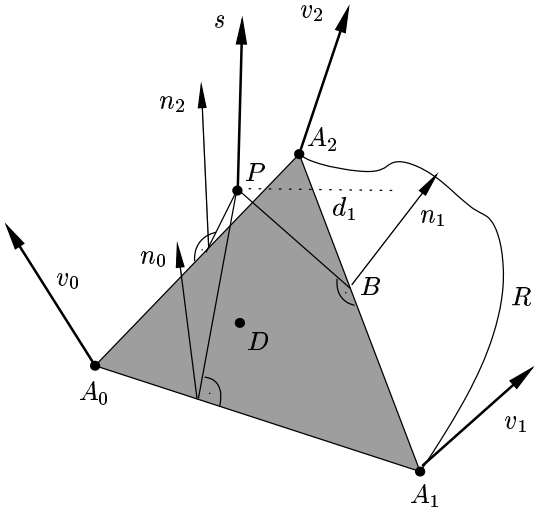

Figure 2: Force calculation.

Algorithm 1. for the small triangle set works as follows: For each triangle (fig.2) we check first if it has the proper orientation and distance with respect to the current position P of PHANTomM. Then we continue with the following step. For each edge of the triangle

Algorithm 1 Force calculation

$force = (0,0,0)$
$maxforce = 0$
for all $T \in object$ **do**
  $d \Leftarrow distance(P,T)$
  if $d > 0$ **and** $d < RADIUS$ **then**
    for $i = 0$ **to** 3 **do**
      $B \Leftarrow nearestpoint(edge(T,i),P)$
      $n_i \Leftarrow interpolation(T,i,B)$
      $R \Leftarrow plane(B,n_i,edge(T,i))$
      $d_i \Leftarrow distance(P,R)$
    end for
    if $d_0 \geq 0$ **and** $d_1 \geq 0$ **and** $d_2 \geq 0$ **then**
      $n_0 \Leftarrow \frac{n_0}{d_0}, n_1 \Leftarrow \frac{n_1}{d_1}, n_2 \Leftarrow \frac{n_2}{d_2}$
      $n \Leftarrow n_0 + n_1 + n_2$
      $D \Leftarrow intersection(n,T)$
      if $|\overrightarrow{DP}| < RADIUS$ **then**
        $s \Leftarrow f(\overrightarrow{DP})$
        $force \Leftarrow force + s$
        $maxforce \Leftarrow max(maxforce,|s|)$
      end if
    end if
  end if
end for
$force = \frac{force}{|force|} * maxforce$

we find the closest point and by linear interpolation we compute the normal vector at this point. Normal vector and edge define plane R. If the point P is bellow this plane we stop processing this triangle. Now we have three normals and distances from planes. Each normal is divided by its distance from the plane R to get proportional force. Summing all three of them we get the total force which is pushing the user away from this triangle. Now we could also compute virtual contact point (VCP) D as the intersection of the line, which goes through the point P in direction of the resulting normal, with the triangle's plane.

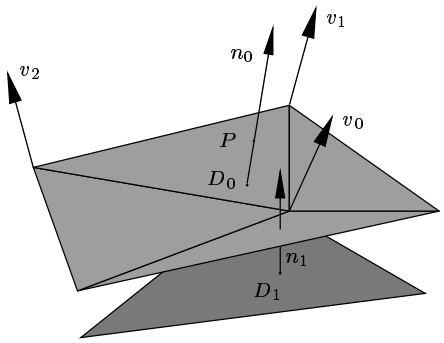

Figure 3: Overlapping triangles.

We resolve all VCPs and we remove those which are actually hidden bellow some other triangles (fig.3). This could be easily achieved just by checking if the line between VCP and real position P is not intersecting another triangle with VCP. Finally we combine all forces together. Resulting force is actually reasonably good approximation which is a compromise between the speed of calculation and the precision of the resulting force.

Now we have smooth normal vectors without any bouncing change in the direction, but we still do not know whether user actually penetrates the surface. To solve this problem we use the following procedure:

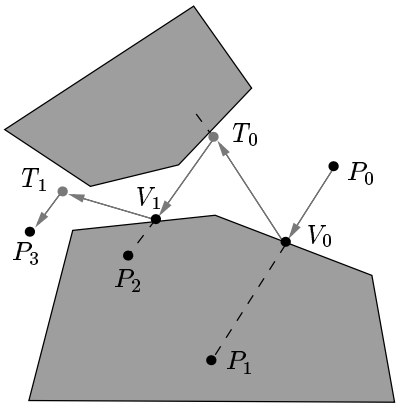

Figure 4: Penetration of the surface.

If during our calculation there is no intersection on path between two consecutive points e.g. P_0 and P_1 (fig.4) with any of processed triangles then the user is not inside the object.

If such point is found then we use the VCP (V_0) for the force calculation. In the following iteration we add difference vector ($\overrightarrow{P_1P_2}$) to V_0 and we try to find new intersection points. We take into account only those triangles where we detect the movement from inside to outside half space. In the case of more than one intersection we pick the closest point (T_0) and use it for calculation of the new VCP (V_1). If there is no such point we use current VCP.

5 Fixed Direction Hulls

To select a small amount of triangles from a subspace we need to use very fast method. For static scenes we have achieved good results with Fixed Direction Hulls also known as k-Dops (Discrete Oriented Polytops) see fig.5. Details can be found in [6],[7].

FDHs have almost the same processing speed as AABBs, but they bound objects more tightly, just like OBBs. An advantage of FDH is lower query time.

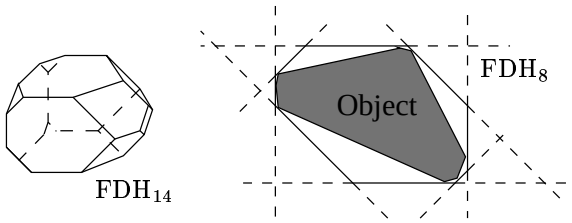

Figure 5: FDH_{14} and 2-D example (FDH_8).

On the other hand their disadvantage is that, there is no simple solution for rebuilding FDH after rotational transformations. Hence we prefer this method for static scenes where only few objects move (user's hand, in our case). We use BVHs which are composed from FDH_{14}. One FDH_{14} is internally represented by 14 floating point numbers which define 14 cutting planes with given normals. The first 6 planes are the same as in AABB. Most of the computations when testing collisions in rarely populated scenes, have the same averrige complexity as when using AABB.

6 Implementation Details

For implementation of this system we developed driver for hard real time OS RT–Linux. We are able to test wide spectrum of update rates. Hence in this area we are exceeding the possibilities of the original software supplied with PHANToM device. While the standard implementation for Window NT operates on 1kHz update rate we are able to use rates up to 10kHz[2] with the possibility to change this speed in runtime.

For our experiments we use dual processor system with two 470MHz Celerons running RT-Linux 2.4.5 and Matrox G400 3D acceleration graphics card.

With our demonstration application we use polygonal models consisting from around 20000[3] triangles. We achieve FDH query time within the range of 10ms and less (typically around 3ms) while the hard real-time loops runs with 64 selected triangles and the system still show around 50% idle time with 3kHz update rate.

7 Conclusions and Future Work

Currently we can process only static scenes, but we prepare implementation of dynamic algorithm which will allow us to update triangle sets and to do a smooth transition between two sets.

[2] according to our experiments [10] this appears to be rather a hardware limit for this device with PCI card.

[3] so they could be displayed with acceptable frame rate on the graphics card

Availability

The latest informations about our software, which is available under GPL, can be found on our web page: http://decibel.fi.muni.cz/phantom/

Acknowledgements

This project is supported by Grant Agency of Czech Republic, Contact No. GAČR 201/98/K041. Author would like to thank P. Konečný and J. Bečvář for their work and ideas which are presented here and also to J. Sochor for his comments and suggestions and all members of our laboratory.

May 2001

References

[1] D.E. Johnson, E. Cohen; An Improved Methods For Haptic Tracking Of A Sculptured Surface; 2000.

[2] A. Gregory, M.C. Lin, S. Gottschalk, R. Taylor; Fast and Accurate Collision Detection For Haptic Interaction Using 3DOF Force–Feedback Device, 2000.

[3] http://graphics.stanford.edu/~lizhang/collision/ /index.html

[4] W.R. Mark S.C. Randolph, M. Finch, J.M. Van Verth, R.M. Taylor; Adding Force Feedback to Graphics Systems: Issues and Solution. *Proc of SIGGRAPH 96, New Orleans, Louisiana*, 1996.

[5] M. Garland; Quadric-Based Polygonal Surface Simplification 1999.

[6] P. Konečný; Bounding Volumes in Computer Graphics. *Master thesis FI MU*, 1998.

[7] J.T. Klosowski; Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments. *Ph.D. Dissertation, State University of New York at Stony Brook*, May 1998.

[8] J. Bečvář; Application of textures in Haptics. *Master thesis FI MU*, 2000.

[9] A.D. Gregory, S.A Ehmann, M.C Lin; inTouch: Interactive Multiresolution Modelling and 3D Painting with a Haptic Interface. 1999.

[10] Z. Kabeláč; Rendering stiff walls with PHANToM. *PUGS2000, Zurich*, 2000.