# GPU based Detection and Mapping of Collisions for Haptic Rendering in Immersive Virtual Reality

Bernhard Spanlang*, Jean-Marie Normand*, Elias Giannopoulos* and Mel Slater*†‡
*EVENTLab, Facultat de Psicologia, Universitat de Barcelona, Spain
†ICREA – Institució Catalana de Recerca i Estudis Avançats, Spain
‡Department of Computer Science, University College London, UK

***Abstract*—We present a method that maps collisions on a dynamic deformable virtual character designed to be used for tactile haptic rendering in Immersive Virtual Reality (IVR). Our method computes exact intersections by relying on the use of programmable graphics hardware. Based on interference tests between deformable meshes (an avatar controlled by a human participant) and a few hundred collider objects, our method gives coherent haptic feedback to the participant. We use GPU textures to map surface regions of the avatar to haptic actuators. We illustrate our approach by using a vest composed of vibrators for haptic rendering and we show that our method achieves collision detection at rates well over 1kHz on good quality deformable avatar meshes which makes our method suitable for video games and virtual training applications**

## I. Introduction

In this paper we propose a precise collision detection technique to control vibrotactile devices for IVR applications. Our method is based on a geometry shader, thus collisions are correctly detected without relying on bounding volumes but based on the exact geometry of the virtual characters. The main benefit of our method from a computational point of view is that by taking advantage of the programmable graphics hardware, our approach is fast, extremely easy to implement and that it is performed in a single render pass. For precise tactile feedback, our system detects exact collisions in the Virtual Environments (VE) based on the deformable geometry of each 3D avatar.

The remainder of this article is structured as follows: the next section is devoted to some background on tactile feedback. Section III presents our GPU based collision detection technique for deformable dynamic characters in VE, Section IV gives performance results while Section V covers the hardware vibrotactile setup we used. Section VI presents how we mapped the collision detection results of our geometry shader to the vibrotactile actuators. Finally we present conclusions and suggest future work.

## II. Background

Lécuyer et al. [1] showed that haptic feedback (rotating the participant's wrist) improves the perception of self-motion in VR, reinforcing the idea that haptic feedback is important to enhance the feeling of presence. In [2], Bloomfield and Badler used vibrotactile feedback in order to enhance Karate training in VR by correcting students' Karate movements using a Tactor suit that uses vibrator arrays.

Haptic feedback for VE can also be applied through pneumatic actuators. TNGames[1] propose the FPSVest, a haptic vest composed of 8 pneumatic cells designed to simulate the direction and force of bullet impacts in a first person shooter game. The University of Pennsylvania developed the Tactile Gaming Vest[2] (TGV) designed for the same purposes as the FPSVest. The TGV uses 4 solenoids actuators on the chest and shoulders and 2 in the back of the vest. Moreover, vibrators are clustered around the shoulders to recreate slashing effects.

LindeMan et al. [3] presented a haptic feedback vest developed for military purposes. It was composed of 16 vibrotactile devices designed to improve soldiers experience during virtual training. As for the other vests, tactors are fired whenever collisions are detected in the virtual world. However, this system only relies on a fairly simple collision detection based on bounding boxes of the skeleton of the soldier's virtual avatar.

Next we introduce our exact, fast and easy to implement GPU based collision detection which is tailored to map collisions to tactile actuators.

## III. GPU based Collision Detection Geometry Shader for Deformable Dynamic Virtual Characters

Methods to detect collisions have been the topic of intensive research over the last three decades mainly for computer graphics and virtual reality applications. Many methods have been introduced that are based on volume hierarchies, in which the volumes range from roughly fitting bounding spheres to tighter fitting axis or object aligned bounding boxes to even closer fitting volumes.

Early approaches mainly aimed at detecting collisions between rigid objects but, with increasing processing power, collisions between deformable objects such as in cloth or soft tissue simulations were introduced. An overview of collision detection with deformable objects on the CPU and GPU is given in [4]. For a detailed and complete survey on recent real-time collision detection methods both on CPU and GPU the reader is invited to refer to [5].

Our method performs an intersection test between every triangle of the deformable meshes and the rays formed by

[1]http://tngames.com/
[2]http://haptics.seas.upenn.edu

978-1-4244-6509-5/10/$26.00 ©2010 IEEE

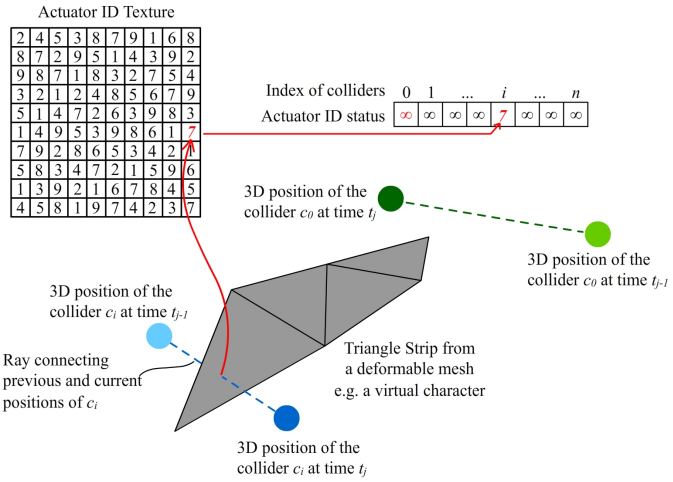


Fig. 1. Illustration of our Geometry shader. When a collision is detected, the actuator ID map is looked-up in order to know which actuator should be triggered for haptic feedback.

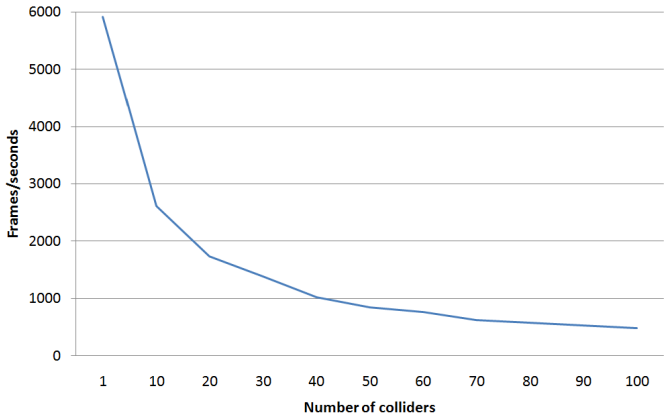


Fig. 2. Collision detection shader performance. One dynamic avatar (∼5000 triangles) is tested against up to 100 dynamic colliders.

the previous and current position of all potentially colliding objects (cf. Fig. 1).

On the CPU such an approach would be highly inefficient and has a complexity of $O(n^2)$. However, because we perform this computation on the GPU's highly parallel geometry shader units it remains efficient enough for haptic rendering (cf. Section IV) even without possible additional optimisation.

Whenever a collision between a triangle of a mesh and a ray (representing the movement of each collider since last frame) is detected, we look-up in the actuator ID texture and emit a vertex from the geometry shader that has as an attribute the ID of the corresponding actuator, cf. Fig. 1. If no collision is detected, no vertex is emitted and nothing is written in the collision texture. As a consequence, fragments corresponding to the colliders that did not collide will contain the clear color value (illustrated by $\infty$ in Fig. 1) since nothing is written into it.

The encoding of the collision information is performed by writing the actuator ID of the colliding point of the avatar surface into the corresponding address in a floating point render target. This requires converting the ID of the collider to the normalized device coordinate (NDC) system for the texture, where both X and Y values are in the interval $[-1, 1]$. We thus map the collider's ID (i.e. a value between 0 and the number of colliders $n_c$) to the interval $[-1, 1]$, collider 0 being mapped to the first pixel (i.e. the pixel corresponding to the NDC -1) while collider $n_c$ corresponds to the last pixel (which NDC coordinate is 1).

In the fragment shader we write the information required for the collision response to the fragment color. In our case, this information corresponds to the ID of the actuator that should be activated in case of collision. Fig. 1 illustrates a collision between collider $c_i$ and a triangle of the mesh that activates vibrator number 7 (since 7 is stored in the actuator texture addressed by the vertex texture coordinates of the triangle).

On recent GPUs we can also store additional information that could be required for the haptic rendering (e.g. speed of the vibrator or pressure of an air cell) in up to 16 additional render targets (depending on the GPU hardware) where each target consists of fragments that contain a vector of 4 floating point values.

## IV. Performance

In this section, we give details about the performance achieved with our geometry shader for collision detection. The system we used to test our method was a Windows 7 32bit machine with 3GBytes of RAM and an nVidia Geforce GTX285 graphics card embedding 1GByte of GDDR3 memory. The virtual characters we used were composed of approximately 3000 vertices (5000 triangles), fully textured and animated. In our tests we used linear blend skinning to deform the avatars' meshes in the GPU's vertex shader according to the skeletal state of the avatar. This was achieved through HALCA [6], a hardware accelerated library for character animation.

The time required for collision detection on a fully animated virtual character with approximately 5k triangles is given in Figures 2. Our test showed that our collision detection runs at over 5kHz for a single collider and at more than 1kHz for up to 40 colliders. These tests were designed to measure how fast the collision detection could be and were performed with gDEBugger GL. This software developed by graphicRemedy[3] can give precise performance measurements on OpenGL applications.

In order to obtain such performance on an application, note that one would have to implement two threads using two separated OpenGL contexts. One of those threads would be dedicated to detecting the collisions, filling the actuator textures within a first OpenGL context and firing the vibrators, while the second thread and OpenGL context would be dedicated to the animation of the avatars and the rendering of the 3D scene.

Our approach is therefore well suited for the haptic feedback purpose in IVR applications which we focus on. Indeed, in

[3]http://www.gremedy.com/

this context the number of characters controlled is likely to be low, allowing us to compute our collision detection efficiently (in less than 2ms, i.e. more than 500Hz). This is for 100 potentially simultaneous colliders which is sufficient for many haptic applications. We haven't measured the latency of the system in detail but owing to our observations it is very low.

## V. Haptic Hardware Setup

The haptic interface used for this system is a haptic vest that we have developed at the EVENTLab. It is composed of a Velcro® vest, an array of six vibrators and a microcontroller board.

The microcontroller board is an Arduino[4] MEGA which utilizes the ATmega1280[5] programmable microcontroller. Furthermore the microcontroller is coupled with an Xbee shield, which allows the microcontroller to communicate wirelessly with the computer, through the Zigbee[6] specification. The microcontroller is programmed to trigger the appropriate vibrators connected through the board's output pins according to the collision detection texture retrieved from the GPU.

The vibrators are coin type vibrators, which are encapsulated within a metal casing, ensuring that no moving parts come in contact with the user's body. The vibrators, used with our haptic vest, operate at a rate of 9000 rpm. The intensity of the vibration can be controlled by modulating the output signal, resulting in a reduced output voltage and thus lower motor speed and smaller vibration. This can be achieved through Arduino's custom Pulse-Width Modulated (PWM) output pins or through any other digital output pins, provided that the microcontroller has been programmed to efficiently modulate the output signal. The vibrators have been mounted on small boards that can be clipped on custom made Velcro® strips, cf. Fig. 3. This way, the vibrators can be placed anywhere on the vest to accommodate a variety of configurations to meet the requirements of the application.

## VI. Mapping Collision Information to Vibrotactile Haptic Rendering

Section III presented our GPU based collision detection algorithm for virtual characters controlled by a user while the previous section detailed the hardware vest we used for haptic rendering. Now we explain how we map collision information and the haptic feedback that should be felt by the participant. The choice of this mapping can be easily modified depending on the experiment carried out and the hardware setup used for haptic rendering.

The mapping is coherent, meaning that the participant will feel haptic feedback on his or her chest if the collisions were detected on the chest of the virtual character in which he or she is embodied. Of course, one can easily imagine a mapping that would give haptic feedback on a different location on the body than where the collision happened in the IVE. Fig. 4

[4]http://www.arduino.cc/
[5]http://www.atmel.com
[6]http://www.arduino.cc/en/Main/ArduinoXbeeShield

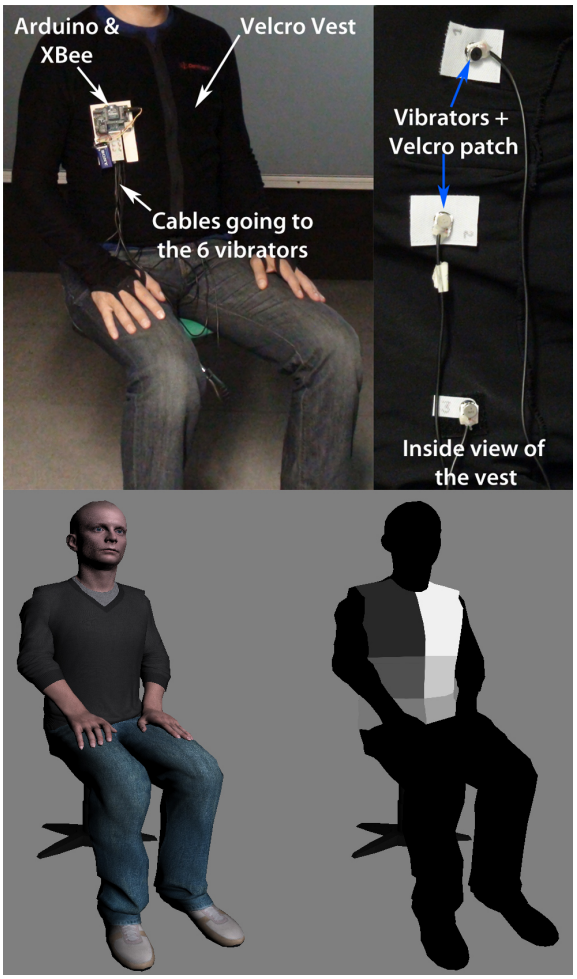


Fig. 3. Haptic rendering setup on a participant (top). Illustration of the corresponding virtual character (bottom left) and of the actuator ID texture used for collision to vibrators mapping (bottom right).

illustrates how we mapped 6 parts of the avatar's torso to 6 different vibrators we used for the haptic feedback.

Each of the 6 gray areas of the torso on the right hand side of Fig. 4 corresponds to a unique vibrator identifier in our haptic vest. Areas outside those 6 colors (i.e. black values in the texture) will not trigger any haptic feedback.

Whenever a collision is detected in the geometry shader, cf. Section III, the texture coordinates of the vertices of the triangle involved is used to look-up the vibrator index in the actuator ID texture (highlighted in Fig. 4, right). To illustrate our method, we only used 6 vibrators; but obviously any number of vibrators that can be uniquely encoded in the texture can be used.

As a consequence, the collision detected on a triangle of the top right part of the chest of the avatar (highlighted in red on left hand side of Fig. 4) will fire vibrator number 1, while a collision detected on a triangle on the bottom left part of the chest of the avatar (cf. blue highlight of Fig. 4 left hand side) will activate vibrator 4. The same technique is applied for each triangle of the avatar.

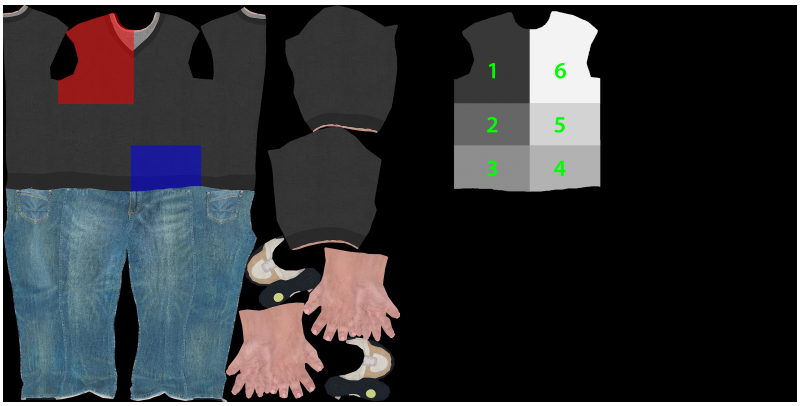


Fig. 4. The actuator ID texture (right) is created in order to map the triangles of the avatar (based on the diffuse map, left) to the vibrators of the haptic vest. Level of gray areas of the actuator map will be mapped to some vibrators, while black areas will not be mapped to any vibrator.

Obviously the same technique can be applied for a much larger number of vibrators that could cover the whole body of the user. Our method could then allow a full-body haptic feedback with collision detection using the exact geometry of the dynamic deformable meshes. The mapping of vibrators to the avatar's body can be easily changed by editing the actuator ID map. This can be achieved either by using image editing software like Photoshop, or if required by programming. Obviously the mapping can be changed also dynamically by modifying the actuator ID map.

## VII. Conclusion & Future Work

In this paper we have presented a novel method to perform exact vibrotactile feedback based on collision detection performed on the GPU. Our method is very flexible and efficient, requires only a single render pass, performs exact collision detections and has no restrictions on the shape of the meshes.

Such an approach could not be performed relying on classical CPU-based collision detection techniques because we do not have direct access to the geometry of the meshes on the CPU if they are deformed by an avatar's skeleton on the GPU. Hence, a similar method on the CPU would be very time consuming because the mesh information would either have to be computed on the CPU or it would have to be transferred from the GPU to the CPU in every animation cycle. Our method has the advantage of being independent of the shape of the model since our geometry shader only relies on triangular meshes. Moreover unlike image based approaches our method is view independent.

The method is most suitable for applications in which meshes are deformed on the GPU in which there are a few, up to several hundred, simple colliders. In order to demonstrate the effectiveness of our approach, we applied it in an IVR application. In this setup, a participant controlled a virtual avatar and was equipped with a vibrotactile haptic vest composed of 6 vibrators. We used our collision detection technique to activate the vibrators whenever a collision appeared on the virtual body of the avatar.

For very large meshes the performance of the collision detection can be improved by pre-culling mesh regions that will definitely not collide, using the traditional bounding volumes on the GPU. Haptic feedback can be improved by storing additional information such as speed of the vibrator, etc. in the texture by using the *gl_FragData* GLSL mechanism in the geometry and fragment shaders in order to take advantage of multiple render targets, which number is GPU dependent. This extra storage space would allow us to store more information on how to perform the haptic rendering (e.g. storing the rotation speed of the vibrators, the pressure of the air in air cells, etc.).

As future work, we plan to focus on improving the way we store information in the collision texture in order to perform smooth transitions from one actuator to another. This could be used for example to perform haptic rendering for when a virtual object slides along the chest of an avatar. Instead of enabling the actuators one after the other, we plan to create the sensation of sliding by varying the actuator intensities.

## Acknowledgment

This work was started under the EU FET project PRESENCCIA, and has continued under the EU FP7 project BEAMING and the ERC project TRAVERSE.

## References

[1] A. Lécuyer, M. Vidal, O. Joly, C. Mégard, and A. Berthoz, "Can haptic feedback improve the perception of self-motion in virtual reality?" *Haptic Interfaces for Virtual Environment and Teleoperator Systems, International Symposium on*, vol. 0, pp. 208–215, 2004.

[2] A. Bloomfield and N. I. Badler, "Virtual training via vibrotactile arrays," *Presence: Teleoper. Virtual Environ.*, vol. 17, no. 2, pp. 103–120, 2008.

[3] R. W. Lindeman, R. Page, Y. Yanagida, and J. L. Sibert, "Towards full-body haptic feedback: the design and deployment of a spatialized vibrotactile feedback system," in *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*. New York, NY, USA: ACM, 2004, pp. 146–149.

[4] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, and W. Strasser, "Collision detection for deformable objects," in *Eurographics State-of-the-Art Report (EG-STAR)*, Eurographics Association. Eurographics Association, 2004, pp. 119–139. [Online]. Available: http://www-evasion.inrialpes.fr/Publications/2004/TKZHRFCFMS04

[5] C. Ericson, *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[6] M. Gillies and B. Spanlang, "Comparing and evaluating real time character engines for virtual environments," *Presence: Teleoper. Virtual Environ.*, vol. 19, no. 2, pp. 95–117, 2010.