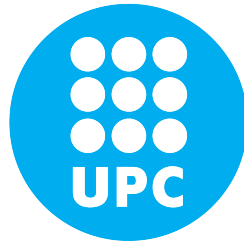


**ADVERTIMENT.** La consulta d'aquesta tesi queda condicionada a l'acceptació de les següents condicions d'ús: La difusió d'aquesta tesi per mitjà del servei TDX ([www.tesisenxarxa.net](http://www.tesisenxarxa.net)) ha estat autoritzada pels titulars dels drets de propietat intel·lectual únicament per a usos privats emmarcats en activitats d'investigació i docència. No s'autoritza la seva reproducció amb finalitats de lucre ni la seva difusió i posada a disposició des d'un lloc aliè al servei TDX. No s'autoritza la presentació del seu contingut en una finestra o marc aliè a TDX (framing). Aquesta reserva de drets afecta tant al resum de presentació de la tesi com als seus continguts. En la utilització o cita de parts de la tesi és obligat indicar el nom de la persona autora.

**ADVERTENCIA.** La consulta de esta tesis queda condicionada a la aceptación de las siguientes condiciones de uso: La difusión de esta tesis por medio del servicio TDR ([www.tesisenred.net](http://www.tesisenred.net)) ha sido autorizada por los titulares de los derechos de propiedad intelectual únicamente para usos privados enmarcados en actividades de investigación y docencia. No se autoriza su reproducción con finalidades de lucro ni su difusión y puesta a disposición desde un sitio ajeno al servicio TDR. No se autoriza la presentación de su contenido en una ventana o marco ajeno a TDR (framing). Esta reserva de derechos afecta tanto al resumen de presentación de la tesis como a sus contenidos. En la utilización o cita de partes de la tesis es obligado indicar el nombre de la persona autora.

**WARNING.** On having consulted this thesis you're accepting the following use conditions: Spreading this thesis by the TDX ([www.tesisenxarxa.net](http://www.tesisenxarxa.net)) service has been authorized by the titular of the intellectual property rights only for private uses placed in investigation and teaching activities. Reproduction with lucrative aims is not authorized neither its spreading and availability from a site foreign to the TDX service. Introducing its content in a window or frame foreign to the TDX service is not authorized (framing). This rights affect to the presentation summary of the thesis as well as to its contents. In the using or citation of parts of the thesis it's obliged to indicate the name of the author

# Enhancing detailed haptic relief for real-time interaction



**Víctor Theoktisto Costa**

ADVISORS: Marta Fairén, Isabel Navazo

Departament de Ciències de la Computació  
Universitat Politècnica de Catalunya  
BarcelonaTECH

**Doctoral Thesis**

*PhD Program in Software*

Barcelona

September 2015



*Not that kind of a Doctor... Penny*

*The Big Bang Theory*

First of all, I wish to dedicate this thesis to my mother Pilar,  
still going strong, and still thinking that what I do is, well, weird.

To my late father Nikita who saw me start in this  
endeavor but sadly didn't get to see it come about.

To Désirée, who has been with me in and out throughout this process.  
Words cannot describe what having her by my side has meant to me.

To my children Víctor Arturo, Jessica Patricia and Lucía Gabriela, who grew amusedly  
watching their father struggling with research, books, and articles as much as they did.  
Los quiero mucho.

To my esteemed colleagues at the Departamento de Computación y Tecnología  
de la Información at the Universidad Simón Bolívar for their support.

To the people of the Moving Research Group at the Universitat Politècnica  
de Catalunya, the CRV/ViRVIG of Barcelona, and in particular to Professor  
Pere Brunet, many thanks for receiving and hosting me all these years.

To my friends Eva Monclús and Jordi Moyés, for their generous help in technical and  
logistical matters. To all my mates at the program, thanks for the camaraderie (beers, too).

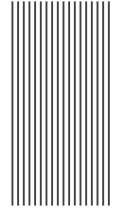
And finally to the most excellent advisors residing this side of the asteroid belt  
Professor Isabel Navazo, for her kind but relentless guidance and steely vision; and  
Professor Marta Fairén, for her nonsense cutting, great advice and enthusiastic collaboration.  
To both of them my deepest thanks to their constant encouragement, prodding and patience.

Above all patience. Lots of it.





## ***Acknowledgements***

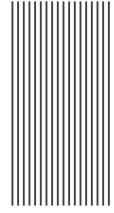


*This work has been partially co-financed by:*

Project TIN2010-20590-C02-01 of the Spanish Government (MEC),  
Project TIN2007-67982-C02-01 of the Spanish Government (MEC),  
Project TIN2004-08065-C02-01 of the Spanish Government (MEC) and FEDER funding,  
Project II-021-FA of the UE ALFA Cordial-2 network.



## *Abstract*



The present document exposes a different approach for haptic rendering, defined as the simulation of force interactions to reproduce the sensation of surface relief in dense models. Current research shows open issues in timely haptic interaction involving large meshes, with several problems affecting performance and fidelity, and without a dominant technique to treat these issues properly.

Relying in pure geometric collisions when rendering highly dense mesh models (hundreds of thousands of triangles) sensibly degrades haptic rates due to the sheer number of collisions that must be tracked between the mesh's faces and a haptic probe. Several bottlenecks were identified in order to enhance haptic performance: software architecture and data structures, collision detection, and accurate rendering of surface relief.

To account for overall software architecture and data structures, it was developed a complete component framework for transforming standalone VR applications into full-fledged multi-threaded Collaborative Virtual Reality Environments (CVREs), after characterizing existing implementations into a feature-rich superset. Enhancements include: a scalable arbitrated peer-to-peer topology for scene sharing; multi-threaded components for graphics rendering, user interaction and network communications; a collaborative user interface model for session handling; and interchangeable user roles with multi-camera perspectives, avatar awareness and shared annotations. We validate the framework by converting the existing ALICE VR Navigator into a complete CVRE, showing good performance in collaborative manipulation of complex models.

To specifically address collision detection computation, we derived a conformal algebra treatment for collisions among points, segments, areas, and volumes, based on collision detection in conformal  $\mathbb{R}^{4,1}$  (5D) space, and implemented in GPU for faster parallel queries. Results show orders of magnitude time reductions in collisions computations, allowing interactive rates.

Finally, the main core of the research is the haptic rendering of surface mesostructure in large meshes. Initially, a method for surface haptic rendering was proposed, using image-based Hybrid Rugosity Mesostructures (HRMs) of per-face heightfield displacements and normalmaps layered on top of a simpler mesh, adding greater surface detail than actually present. Haptic perception is achieved modulating the haptic probe's force response using the HRM coat. A usability testbed framework was built to measure experimental perfor-

mance with a common set tests, meshes and HRMs. Trial results show the goodness of the proposed technique, rendering accurate 3D surface detail at high sampling rates.

This local per-face method is extended into a fast global approach for haptic rendering, building a mesostructure-based atlas of depth/normal textures (HyRMA), computed out of surface differences of the same mesh object at two different resolutions: original and simplified. For each triangle in the simplified mesh, an irregular prism is considered defined by the triangle's vertices and their normals. This prism completely covers the original mesh relief over the triangle. Depth distances and surfaces normals within each prism are warped from object volume space to orthogonal tangent space, by means of a novel and fast method for computing barycentric coordinates at the prism, and storing normals and relief in a sorted atlas. Haptic rendering is effected by colliding the probe against the atlas, and effecting a modulated force response at the haptic probe. The method is validated numerically, statistically and perceptually in user testing controlled trials, achieving accurate haptic sensation of large meshes' fine features at interactive rendering rates, with some minute loss of mesostructure detail.

## *Resum*

En aquesta tesi es presenta un novedós enfocament per a la percepció hàptica del relleu de models virtuals complexos mitjançant la simulació de les forces d'interacció entre la superfície i un element de contacte. La proposta contribueix a l'estat de l'art de la recerca en aquesta àrea incrementant l'eficiència i la fidelitat de la interacció hàptica amb grans malles de triangles.

La detecció de col·lisions amb malles geomètriques denses (centenars de milers de triangles) limita la velocitat de resposta hàptica degut al gran nombre d'avaluacions d'intersecció cara-dispositiu hàptic que s'han de realitzar. Es van identificar diferents alternatives per a incrementar el rendiment hàptic: arquitectures de software i estructures de dades específiques, algorismes de detecció de col·lisions i reproducció hàptica de relleu superficial. En aquesta tesi es presenten contribucions en alguns d'aquests aspectes.

S'ha proposat una estructura completa de components per a transformar aplicacions aïllades de Realitat Virtual en Ambients Col·laboratius de Realitat Virtual (CRVEs) multithread en xarxa. L'arquitectura proposada inclou: una topologia escalable punt a punt per a compartir escenes; components multithread per a visualització gràfica, interacció amb usuaris i comunicació en xarxa; un model d'interfície d'usuari col·laboratiu per a la gestió de sessions; i rols intercanviables de l'usuari amb perspectives de múltiples càmeres, presència d'avatars i anotacions compartides. L'estructura s'ha validat convertint el navegador ALICE en un CVRE completament funcional, mostrant un bon rendiment en la manipulació col·laborativa de models complexos.

Per a incrementar l'eficiència del càlcul de col·lisions, s'ha proposat un algorisme que treballa en un espai conforme  $\mathbb{R}^{4,1}$  (5D) que permet detectar col·lisions entre punts, segments, triangles i volums. Aquest algorisme s'ha implementat en GPU per tal d'obtenir una execució paral·lela més ràpida. Els resultats mostren reduccions en el temps de càlcul de col·lisions permetent resposta interactiva.

Per a la percepció hàptica de malles complexos que modelen objectes rugosos, s'han proposat diferents algorismes i estructures de dades. Les denominades Mesoestructures Híbrides de Rugositat (HRM) permeten substituir els detalls geomètrics d'una cara (rugositats) per una textura de normals i una altra d'alçades. La percepció hàptica s'aconsegueix modulant la força de resposta entre el dispositiu hàptic i la HRM. Els tests realitzats per

avaluar experimentalment l'eficiència del càlcul de col·lisions i la percepció hàptica utilitzant HRM respecte a modelar les rugositats amb geometria, van mostrar que la tècnica proposada va ser encertada, permetent percebre detalls de superfície 3D correctes a altes taxes de mostreig.

El mètode anterior es va estendre per a representar rugositats d'objectes. Per a fer-ho es proposa substituir l'objecte per un model simplificat i un atlas de mesoestructures en el que s'usen textures de normals i de relleus (HyRMA). Aquest atlas s'obté a partir de la diferència en el detall de la superfície entre dos malles del mateix objecte: l'original i la simplificada. A partir d'un triangle de la malla simplificada es construeix un prisma, definit pels vèrtexs del triangle i les seves normals, que engloba completament el relleu de la malla original sobre aquest triangle. Les alçades i normals dins de cada prisma es transformen des de l'espai de volum a l'espai ortogonal tangent, usant un mètode novedós i eficient que calcula les coordenades baricèntriques relatives al prisma, per a guardar el mapa de textures transformat en un atlas ordenat. La percepció hàptica s'aconsegueix detectant directament les col·lisions entre el dispositiu hàptic i l'atlas, i modulant la força de resposta d'acord al resultat de la col·lisió. El mètode s'ha validat numèricament, estadísticament i perceptualment en tests amb usuaris, aconseguint una correcta i interactiva sensació tàctil dels objectes simulats mitjançant la mesoestructura de les malles, amb una pèrdua molt menor de detall.

## *Resumen*

En esta tesis se presenta un enfoque novedoso para la percepción háptica del relieve de modelos virtuales complejos mediante la simulación de las fuerzas de interacción entre la superficie y un elemento de contacto. La propuesta contribuye al estado del arte de investigación en este área incrementando la eficiencia y fidelidad de interacción háptica con grandes mallas de triángulos.

La detección de colisiones con mallas geométricas densas (cientos de miles de triángulos) limita la velocidad de respuesta háptica debido al elevado número de evaluaciones de intersección cara-dispositivo háptico que deben realizarse. Se identificaron diferentes alternativas para incrementar el rendimiento háptico: arquitecturas de software y estructuras de datos específicas, algoritmos de detección de colisiones y reproducción háptica de relieve superficial. En esta tesis se presentan contribuciones en algunos de estos aspectos.

Se ha propuesto una estructura completa de componentes para transformar aplicaciones aisladas de Realidad Virtual en Ambientes Colaborativos de Realidad Virtual (CRVEs) multithread en red. La arquitectura propuesta incluye: una topología escalable punto a punto para compartir escenas; componentes multithread para visualización gráfica, interacción con usuarios y comunicación en red; un modelo de interfaz de usuario colaborativo para la gestión de sesiones; y roles intercambiables del usuario con perspectivas de múltiples cámaras, presencia de avatares y anotaciones compartidas. La estructura se ha validado convirtiendo el navegador ALICE en un CVRE completamente funcional, mostrando un buen rendimiento en la manipulación colaborativa de modelos complejos.

Para incrementar la eficiencia del cálculo de colisiones, se ha propuesto un algoritmo que trabaja en un espacio conforme  $\mathbb{R}^{4,1}$  (5D) que permite detectar colisiones entre puntos, segmentos, triángulos y volúmenes. Este algoritmo se ha implementado en GPU a efectos de obtener una ejecución paralela más rápida. Los resultados muestran reducciones en el tiempo de cálculo de colisiones permitiendo respuesta interactiva.

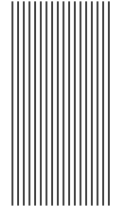
Para la percepción háptica de mallas complejas que modelan objetos rugosos, se han propuesto diferentes algoritmos y estructuras de datos. Las denominadas Mesoestructuras Híbridas de Rugosidad (HRM) permiten substituir los detalles geométricos de una cara (rugosidades) por una textura de normales y otra de alturas. La percepción háptica se consigue modulando la fuerza de respuesta entre el dispositivo háptico y la HRM. Los tests realiza-



dos para evaluar experimentalmente la eficiencia del cálculo de colisiones y la percepción háptica utilizando HRM respecto a modelar las rugosidades con geometría, mostraron que la técnica propuesta fue acertada, permitiendo percibir detalles 3D correctos a altas tasas de muestreo.

Este método anterior es extendido a un procedimiento global para representar rugosidades de objetos. Para hacerlo se propone sustituir el objeto por un modelo simplificado y un atlas de mesoestructuras usando texturas de normales y relieves (HyRMA). Este atlas se obtiene de la diferencia en detalle de superficie entre dos mallas del mismo objeto: la original y la simplificada. A partir de un triángulo de la malla simplificada se construye un prisma definido por los vértices del triángulo a lo largo de sus normales, que engloba completamente el relieve de la malla original sobre este triángulo. Las alturas y normales dentro de cada prisma se transforman del espacio de volumen al espacio ortogonal tangente, usando un método novedoso y eficiente que calcula las coordenadas baricéntricas relativas a cada prisma para guardar el mapa de texturas transformado en un atlas ordenado. La percepción háptica se consigue detectando directamente las colisiones entre el dispositivo háptico y el atlas, y modulando la fuerza de respuesta de acuerdo al resultado de la colisión. El procedimiento se ha validado numérica, estadística y perceptualmente en ensayos con usuarios, consiguiendo a tasas interactivas la correcta sensación táctil de los objetos simulados mediante la mesoestructura de las mallas, con alguna pérdida muy puntual de detalle.

## *Declaration*



I herewith declare that I have produced this document without the undue assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This document has not previously been presented in identical or similar form to any other examination board.

Barcelona, September 2015

Declaro que he redactat aquest document sense l'assistència indeguda de tercers persones i sense fer ús d'instruments diferents als especificats; les nocions preses directa o indirectament d'altres fonts han estat identificades en el text. Aquest document no s'ha presentat en forma idèntica o similar a cap altre ens examinador.

Barcelona, Setembre 2015

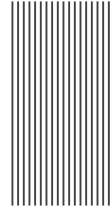
Declaro que he redactado este documento sin la asistencia indebida de terceras personas y sin hacer uso de instrumentos distintos a los especificados; las nociones tomadas directa o indirectamente de otras fuentes han sido identificadas en el texto. Este documento no ha sido presentado en forma idéntica o similar a ningún otro ente examinador.

Barcelona, Septiembre 2015

Víctor Theoktisto Costa  
September 2015



# Contents



<b>Contents</b>	<b>xv</b>
<b>List of Figures</b>	<b>xxi</b>
<b>List of Tables</b>	<b>xxiii</b>
<b>List of Algorithms</b>	<b>xxv</b>
<b>List of Kernel Code Fragments</b>	<b>xxvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 General Motivation . . . . .	1
1.2 Thesis Problem Statement . . . . .	2
1.3 Objective . . . . .	2
1.4 Organization of this document and its contributions . . . . .	2
1.4.1 Review of related works . . . . .	3
1.4.2 Collaborative Virtual Reality Environment Framework . . . . .	3
1.4.2.1 Contributed publications from this area . . . . .	3
1.4.3 Per-face Image-based Haptic Rendering . . . . .	4
1.4.3.1 Contributed publications from this area . . . . .	4
1.4.4 Collision Detection . . . . .	5
1.4.4.1 Contributed publications from this area . . . . .	5
1.4.5 Haptic Rendering using Tangent-space Atlases . . . . .	5
1.4.5.1 Contributed publications from this area . . . . .	6
1.4.6 Discussion . . . . .	7
<b>2 Related work</b>	<b>9</b>
2.1 Collaborative Virtual Environments (CVEs) . . . . .	9
2.1.1 Copresence as shared sensations . . . . .	10
2.1.2 Copresence CVE implementations . . . . .	10
2.1.3 Massively Multiplayer Online Role Playing game engines . . . . .	12
2.2 Collision detection computation strategies . . . . .	13
2.2.1 GPU general computing . . . . .	14

2.2.2	GPU-assisted parallel intersection computation . . . . .	14
2.3	Haptic rendering . . . . .	16
2.3.1	Taxonomies for haptic interaction . . . . .	18
2.3.1.1	Haptic hardware . . . . .	18
2.3.1.2	Haptic detail modeling . . . . .	18
2.3.1.3	Haptic Rendering Algorithms . . . . .	19
2.3.1.4	Haptic space navigation . . . . .	19
2.3.1.5	Haptic multiuser collaboration . . . . .	19
2.3.1.6	Haptic Perception . . . . .	20
2.3.1.7	Haptic user interfaces . . . . .	20
2.3.2	Haptic performance metrics . . . . .	20
2.4	Image-based haptic approaches	
	for geometry substitution . . . . .	21
2.4.1	Deformation and warping . . . . .	21
2.4.2	Texture atlases approaches . . . . .	21
2.4.3	Pixel-based relief rendering . . . . .	22
<b>3</b>	<b>Embedding Collaboration for Virtual Reality</b>	<b>25</b>
3.1	Collaborative Virtual Reality Environments (CVREs) . . . . .	26
3.1.1	Taxonomy of collaboration in virtual reality environments . . . . .	26
3.1.2	Characterization of collaboration in virtual reality environments . . . . .	28
3.2	Collaboration framework architecture . . . . .	29
3.2.1	Multithreaded software components . . . . .	32
3.2.2	P2P sharing topology . . . . .	33
3.2.2.1	Thin broker for session administration . . . . .	34
3.2.2.2	Message protocol . . . . .	34
3.2.3	Session Awareness Management . . . . .	35
3.2.3.1	Collaborative user interface model . . . . .	36
3.2.3.2	Client awareness using <i>avatars</i> . . . . .	36
3.2.3.3	Session management with differentiated session roles . . . . .	37
3.2.3.4	Shared annotation and 3D marker highlighting . . . . .	38
3.2.3.5	External real-time verbal communication channel . . . . .	38
3.2.4	Cross-platform network transmission . . . . .	38
3.2.4.1	System synchronization . . . . .	38
3.3	The ALICE Virtual Reality navigator . . . . .	39
3.4	Performance analysis of the framework . . . . .	40
3.4.1	Framework validation in ALICE . . . . .	40

3.4.1.1	Peer and Broker instantiation . . . . .	41
3.4.1.2	Message protocol implementation . . . . .	41
3.4.2	Emerging ALICE collaborative features . . . . .	42
3.4.2.1	Remote user windows . . . . .	42
3.4.2.2	Extended callback mechanism . . . . .	42
3.4.2.3	Collaborative object manipulation . . . . .	43
3.4.3	Evaluation of results . . . . .	44
3.5	Conclusions from this chapter . . . . .	46
<b>4</b>	<b>Per-face heightfield haptic rendering</b>	<b>49</b>
4.1	Models for haptic perception of surface details . . . . .	49
4.1.1	Generation of geometry . . . . .	50
4.1.2	Force mapping . . . . .	50
4.2	Per-face heightfield haptic rendering algorithm . . . . .	51
4.2.1	Comparison between <i>force mapping</i> and <i>per-face heightfield haptic</i> rendering algorithms . . . . .	53
4.3	Mesostructure model for haptic rendering . . . . .	57
4.3.1	Blending haptic mesostructure at the edges . . . . .	57
4.3.2	Haptic rendering in concave faces . . . . .	59
4.4	Testing Procedure . . . . .	60
4.4.1	Setup . . . . .	60
4.4.2	Test 0 – Baseline perception (Control) . . . . .	62
4.4.2.1	Evaluation of results from Test 0 . . . . .	62
4.4.3	Test I – Quality of Visual-Haptic perception . . . . .	62
4.4.3.1	Evaluation of results from Test I . . . . .	63
4.4.4	Test II – Perception of mesostructure with simple repeating patterns . . . . .	63
4.4.4.1	Evaluation of results from Test II . . . . .	64
4.4.5	Test III – Perception of non-monotonous mesostructure . . . . .	64
4.4.5.1	Evaluation of results from Test III . . . . .	64
4.4.6	Test IV – Perception of visual-haptic disparity in a gradated mesh . . .	66
4.4.6.1	Evaluation of results from Test IV . . . . .	66
4.5	Conclusions from this chapter . . . . .	67
<b>5</b>	<b>Collision detection in conformal space</b>	<b>69</b>
5.1	Geometric Algebra . . . . .	69
5.1.1	Conformal Geometry . . . . .	70

5.1.2	Intersections in conformal space . . . . .	70
5.2	Kernels for intersection algorithms . . . . .	72
5.2.1	Line Segment (Ray)–Triangle (Plane) intersection . . . . .	73
5.2.2	Triangle (Plane)–Triangle (Plane) intersection . . . . .	73
5.2.3	Sphere–Sphere intersection . . . . .	74
5.3	CUDA implementation and results . . . . .	75
5.3.1	Line Segment-Mesh collisions . . . . .	76
5.3.2	$\mathbb{R}^{4,1}$ (CPU 5D) vs. $\mathbb{R}^3$ (CPU 3D) vs $\mathbb{R}^{4,1}$ (GPU 5D) collisions . . . . .	76
5.3.3	Mesh-Mesh collisions . . . . .	78
5.3.4	Sphere – Sphere collisions . . . . .	79
5.4	Conclusions from this chapter . . . . .	82
<b>6</b>	<b>Tangent-space haptic atlases</b>	<b>83</b>
6.1	Tangent-space mesostructure atlas generation . . . . .	83
6.1.1	Per-face normal-depth atlases . . . . .	83
6.1.2	Global-to-local per-face volume warping . . . . .	84
6.1.3	Building parametrically equivalent prismoids . . . . .	84
6.1.4	Obtaining barycentric coordinates applying the “ <i>perp</i> ” operator . . . . .	85
6.2	Tangent-space texture atlas generation using GPU . . . . .	88
6.2.1	Atlas generation procedure . . . . .	89
6.3	Haptic rendering using the tangent-space atlas . . . . .	89
6.3.1	Rendering with a modified Parallax Occlusion Mapping shader . . . . .	91
6.4	Validation and testing procedure . . . . .	93
6.4.1	Setup . . . . .	93
6.4.2	Test I - Trajectory sampling verification . . . . .	95
6.4.2.1	Evaluation of results from Test I . . . . .	96
6.4.3	Test II - Haptic atlas rendering metrics . . . . .	97
6.4.3.1	Evaluation of results from Test II . . . . .	98
6.4.4	Test III - User experience and usability testing . . . . .	99
6.4.4.1	Questionnaire A: Protocol for haptic atlas rendering . . . . .	99
6.4.4.2	Questionnaire B: User perception testing . . . . .	99
6.4.4.3	Evaluation of results from Test III . . . . .	100
6.5	Conclusions from this chapter . . . . .	100
<b>7</b>	<b>Discussion</b>	<b>103</b>
7.1	Summary of conclusions . . . . .	103
7.1.1	Collaborative Virtual Reality Environments . . . . .	103

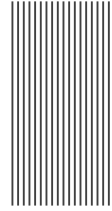
7.1.2	Collision Detection's implementation . . . . .	104
7.1.3	Tangent Space Texture Atlases for haptic rendering . . . . .	104
7.2	Future work . . . . .	105

<b>Bibliography</b>	<b>107</b>
---------------------	------------





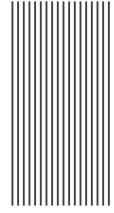
## List of Figures



3.1	Components of Collaborative Virtual Reality Environments. The items placed at the upper half of the circle are those directly related to collaboration issues.	29
3.2	Collaboration-enabling threaded processes. The framework includes original components (GR and UI), and adds a session layer (SS) with the network component (NC).	33
3.3	Peer-to-Peer Broker class model. Both peers and broker have proxy instances of each other.	35
3.4	Model-View-Controller-Session (MVCS) objects showing an external broker maintaining session states.	37
3.5	Two <i>Peers</i> collaborate with each other while navigating the architectural environment, as seen from an invisible <i>incognito</i> client's viewpoint.	40
3.6	A main window with a remote client visualization on the upper-right corner.	44
3.7	Some of the different VR systems (commercial and homegrown) where ALICE has been tested.	45
3.8	Outside and inside views of the Texaco tanker model used in the performance measurements.	46
4.1	Different approaches to simulate surface details in haptic perception.	50
4.2	Correspondence between simulated surface (top) and normal vectors (bottom).	51
4.3	Heightfield collision mapping: once the probe is within a prism, the force's repulsion normal is sampled relative to position, and the force magnitude response is proportional to the penetration.	54
4.4	Normal and height maps of choice test surfaces.	55
4.5	Force-mapping vs. heightfield haptic collisions.	56
4.6	Placing <i>per-face</i> HRM textures (heightfield and normalmaps) on top of triangles. To avoid abrupt surface changes crossing surfaces, a blending function is applied around a ( $\rho$ ) band along edges. Interpolation can be either linear (i), smoothed (ii) or none (iii).	59
4.7	Haptic rendering in concave faces.	60
4.8	Perception scaling adjustment for mesostructure.	63

4.9	Haptic perception of surface frequencies at medium & high resolutions. . . .	65
5.1	Bunny Mesh – Line Segment Intersection. Colliding polygons shown in yellow.	76
5.2	Bunny Mesh – Line Segment Intersection times. . . . .	77
5.3	Fast CPU vs. Conformal CPU Mesh-Triangle Intersections . . . . .	78
5.4	Plane (Triangle) – Bunny mesh Intersection: Colliding polygons shown in yellow. . . . .	79
5.5	Bunny mesh - Armadillo Intersection times. . . . .	80
5.6	Bunny mesh - Armadillo mesh Intersection. Colliding polygons shown in yellow (bunny) and purple (armadillo). . . . .	80
5.7	Spheres - Spheres Intersection. . . . .	81
5.8	Spheres - Spheres Intersection times. . . . .	81
6.1	Heightfield Displacement Computation: Obtain transformations $W_i$ , so each vertex $H_i$ in the $M_F$ mesh that is either within the face's prism or immediately adjacent to such a vertex, is transformed into the orthogonal prism. . . . .	86
6.2	Complete sequence representation of tangent mesostructure atlas construction and image-based relief generation for visualization and real-time haptic rendering. . . . .	88
6.3	Tangent texture atlas for the Stanford Bunny, showing ordering by edge length and alternating up and down orientation. . . . .	90
6.4	Explanation of potential lossy reconstruction using the tangent atlas warp. . .	92
6.5	Parallax Occlusion Mapping shader fundamentals: the View Ray is transformed into the space of the orthogonal prism, and then sampled repeatedly across the (warped) tangent texture space [blue circles]. Two Ray-heightfield intersections [red circles] are shown; only the frontmost collision is accepted, since it occludes the farther ones. . . . .	93
6.6	Trajectory verification by sampling: A path (in red) intersects mesh $B_{256}$ (left), then points along this path are sampled in the HyRMA texture to obtain a reconstructed trajectory, which later is compared to corresponding points sampled and warped from the same path intersecting dense mesh $B_Z$ (right). . . .	96
6.7	Accuracy measurement of height reconstruction. Relative height differences get smaller as the dense mesh's resolution increases, and also do outliers. . .	97
6.8	Haptic framerates for meshes growing in complexity. Performance decrease slowly as mesh complexity grows, but still allowing good interactive rates. . .	98

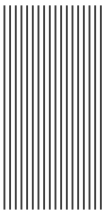
## *List of Tables*



3.1	Characterization of Collaboration Features in CVREs . . . . .	30
3.2	Global Profile of the Studied CVRE's . . . . .	31
3.3	Message Parsing Disassembly . . . . .	35
3.4	Bit code for role identification . . . . .	37
3.5	Collaborative scene performance measurements . . . . .	45
4.1	Trial model meshes. . . . .	61
4.2	Hybrid rugosity mesostructures. . . . .	61
4.3	Trial tests protocol. . . . .	62
4.4	Haptic perception of fine features in non-monotonous mesostructure . . . . .	66
5.1	The 32 (1+5+10+10+5+1) terms of the canonical base for the conformal $\mathbb{R}^{4,1}$ space. . . . .	71
5.2	Algebraic primitives built from blades in the $\mathbb{R}^{4,1}$ conformal space, which also includes scalars, points and vectors . . . . .	72
5.3	Primitive intersections in $\mathbb{R}^{4,1}$ conformal space . . . . .	72
5.4	CPU 5D – GPU 5D performance evaluation. . . . .	76
5.5	GPU 3D collision benchmark times. . . . .	78
5.6	GPU 3D collision benchmark times . . . . .	79
6.1	Test HiRes and LoRes meshes with visual framerates . . . . .	94
6.2	Trajectory verification for the Bunny meshes . . . . .	97
6.3	Haptic Framerate measurements ( <i>in Hits/Sec</i> ) . . . . .	98
6.4	Haptic Atlas rendering measurements . . . . .	99
6.5	User perception of haptic disparity . . . . .	100



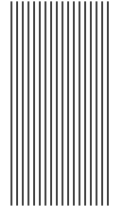
# List of Algorithms



4.1	Haptic force mapping. . . . .	51
4.2	Heightfield-displacement rendering. . . . .	53
4.3	Haptic mesostructure-blended rendering. . . . .	58
6.1	Haptic rendering with HyRMA Atlas . . . . .	91



## *List of Kernel Code Fragments*



5.1	Line Segment-Triangle intersection . . . . .	74
5.2	Triangle-Triangle intersection . . . . .	75
5.3	Sphere-Sphere intersection . . . . .	75





# Introduction



*Everything starts somewhere,  
although many physicists disagree.  
Terry Pratchett*

**T**His chapter introduces the motivation that gave rise to the present work, enunciating the formulated problem statement, the main objective[s] driving the research, a summary of obtained results, and the list of papers detailing the contributions arising from this endeavor.

## 1.1 General Motivation

Haptic perception is a general term denoting the ability to sense variations in geometry, roughness, texture and other volume or surface detail by touch alone, allowing immediate bidirectional interaction between humans and computer-generated objects placed in virtual environments.

Haptic rendering, defined as the simulation of force interactions to reproduce the sensation of surface relief in dense models, requires of special data structures and force-feedback devices having optimal sampling rates of 1000+ Hz to produce accurate touch sensation for model perception. High frequencies are necessary to get closer to natural physical levels of manipulation, necessary in fields such as nanomaterials handling, surgical training, virtual prototyping, machine assembly and digital sculpting. Current research shows open issues in timely haptic interaction involving large models, with several problems affecting performance and fidelity, and without a dominant technique to treat these issues properly.

Relying in pure geometric collisions when rendering highly dense mesh models (ranging from hundreds of thousands to several millions of triangles) sensibly degrades the optimal haptic rates, caused by the sheer number of collisions and force interactions that must be tracked and computed between the mesh's triangle faces and a haptic probe. These identified bottlenecks need to be addressed in order to ensure non-degraded/enhanced haptic performance: overall software architecture and data structures, efficient collision detection, and accurate haptic rendering of surface relief.

## 1.2 Thesis Problem Statement

The purpose of this thesis is to enhance the resulting user perception experience when haptically rendering surface details of highly complex models. We will address this by postulating coupled strategies to ensure “as fast as possible” accurate sensing of haptic generated detail. In particular, a threaded software architecture decoupling image and haptic rendering, fast parallel collision detection, efficient data structures and speedy haptic rendering algorithms that take advantage of all of the above, substituting the objects’ geometry by an image-based approach which encodes geometric surface volume detail in a special tangent space flat texture containing relief and surface normals values. The resulting approach will allow haptical rendering of very dense geometric models in real time, without incurring in performance penalties that might degrade the touch sensation in overall user experience.

## 1.3 Objective

The objective of the research is to develop strategies for a hybrid image-based haptic rendering approach for fast and accurate perception of surface features, ranging from fine creases to major topographic features, and providing a measure of performance against the existing geometric-based haptic rendering algorithms.

The general idea, as in visualization, would be the simulation of roughness and other surface features without having to increase the geometric density of the model, and compare the results of applying both techniques to the same model.

## 1.4 Organization of this document and its contributions

The document has been divided into chapters, each one treating its own research topic separately for better comprehension of the thesis development process.

We will now describe in chronological sequence the main contributions of our research in the following areas: pertinent review of past literature; collaborative virtual reality environment framework; per-face image-based haptic rendering; collision detection in conformal space; and haptic rendering using tangent-space atlases of displacement textures and normalmaps in substitution of actual geometry.

### 1.4.1 Review of related works

Chapter 2 recognizes and evaluates important contributions in the separate areas covered by this research. It is organized by topic, with a commentary on their relevance and suitability, highlighting the issues they do (or do not) address. A conclusion of this review is that the current state of the art provides a space in which the presented work has the potential of adding knowledge and know-how to the chosen fields.

### 1.4.2 Collaborative Virtual Reality Environment Framework

Chapter 3 develops an emerging taxonomy for the characterization of Collaborative Virtual Environments, which is applied to design an add-on scalable component framework that easily incorporates emerging collaboration features to stand-alone Virtual Reality Applications.

Starting from a software architecture perspective, known Collaborative Virtual Reality Environments (CVREs) were studied to devise a proper taxonomical characterization table from the meanings and subtypes of copresence that structure human interaction in virtual environments.

Based on the described collaborative features we developed a “snap-on” superset framework for evolving complete Collaborative Virtual Reality Environments (CVREs) out of existing standalone VR applications. The main result of our approach is a multithreaded architecture with a scalable peer-to-peer network topology that incorporates session layer management, a crossplatform message-passing communications library, and a hybrid collaborative interaction model with multiple avatar roles. The framework adjusts easily to working VR tools without affecting graphics performance.

#### 1.4.2.1 Contributed publications from this area

THEOKTISTO, V., FAIRÉN, M., AND NVAZO, I. (2003). Collaborative virtual reality in the ALICE platform. In *Proceedings of the Spanish Conference in Computer Graphics (CEIG 2003)*, pages 261–276.

THEOKTISTO, V., FAIRÉN, M., AND NVAZO, I. (2004A). ALICE: A Collaborative Virtual Reality Navigator. In *Mendoza, C. and Ganovelli, E, editors, VRIPHYS'04, Workshop on Virtual Reality Interaction and Physical Simulation*, pages 159–169, Colima, Mexico. SMCC (Mexican Society of Computer Science).

THEOKTISTO, V. AND FAIRÉN, M. (2004). On Extending Collaboration in Virtual Reality En-

vironments. In *de Albuquerque Araújo, A., Comba, J. L. D., Navazo, I., and Souza, A. A., editors, Proceedings of the IEEE SIBGRAPI/SIACG'04, II Iberoamerican Symposium on Computer Graphics 2004*, pages 324–331, Curitiba, Brasil. IEEE Computer Society Order Number P2227. ISBN 0-7695-2227-0, ISSN 1530-1834.

Theoktisto, V. and Fairén, M. (2005). Enhancing Collaboration in Virtual Reality Applications. *Computers & Graphics*, 29(5):704–718.

### 1.4.3 Per-face Image-based Haptic Rendering

Chapter 4 proposes a faster method for surface haptic rendering using per-face image-based Hybrid Rugosity Mesostructures (HRMs), with usability trial results and user testing evaluations.

For the haptic part, we developed first a local per-face image-based rendering approach for substituting geometric detail. This method however, although accurate and fast for large planar surfaces, has several shortcomings across mesh edges in non-planar meshes, requiring smoothing functions for stitching displacement textures across edges. This first approach's goals were:

- (i) A specific model and algorithm for rendering image-based mesostructure surface details, mapping dual displacement and normal maps onto underlying simplified geometries (section 4.3);
- (ii) A blending function for smoothing height/normal computation at folding edges and mesostructure transitions (subsections 4.3.1 and 4.3.2); and
- (iii) A battery of usability benchmark tests over a chosen set of meshes and mesostructures, allowing qualitative measures of feature perception at varying resolutions (section 4.4).

Using the previous experimental testing protocol, we achieve accurate haptic perception of fine surface detail without compromising rendering rates or fidelity of touch, with the stated objective of rendering complex detail not present in the original geometry at very low processing costs.

#### 1.4.3.1 Contributed publications from this area

THEOKTISTO, V., FAIRÉN, M., NAVAZO, I., AND MONCLÚS, E. (2005). Rendering detailed haptic textures. In *Proceedings of Second Workshop in Virtual Reality Interactions and Physical Simulations (VRIPHYS '05)*, Pisa, Italy.

THEOKTISTO, V., FAIRÉN, M., AND NAVAZO, I. (2009B). Hybrid rugosity mesostructures (HRMs) for fast and accurate rendering of fine haptic detail. In *Proceedings of XXXV Latin American Computing Conference (CLEI 2009)*, Pelotas, Rio Grande do Sul, Brazil.

THEOKTISTO, V., FAIRÉN, M., AND NAVAZO, I. (2010). A hybrid rugosity mesostructure (HRM) for rendering fine haptic detail. *CLEI Electronic Journal (CLEIej)* ISSN 0717-5000, 13(3):1–12.

#### 1.4.4 Collision Detection

Chapter 5 presents an unified geometric algebra treatment that shifts collision detection from euclidean  $\mathbb{R}^3$  space to a higher conformal  $\mathbb{R}^{4,1}$  space, providing an elegant abstract level to compute collisions among points, vectors, areas and other solid objects. It was developed as a SIMD GPU (CUDA) implementation to account for the increase in dimensionality, providing close to optimal interactive rates.

Results show expected interactive rates improvements when computing collisions and intrusions among known mesh models, providing close to optimal framerate for standard collision implementations, without using any hierarchical bounding volumes or other pre-filtering schemes

##### 1.4.4.1 Contributed publications from this area

ROA, E., THEOKTISTO, V., FAIRÉN, M., AND NAVAZO, I. (2011). GPU Collision Detection in Conformal Geometric Space. In *Proceedings of SIACG 2011: Iberoamerican Symposium on Computer Graphics*, pages 153–156. EuroGraphics.

ROA, E. AND THEOKTISTO, V. (2012). Primitives Intersection with Conformal 5D Geometry. In *Proceedings of CIMENICS 2012, IX International Congress on Numerical Methods in Engineering and Applied Sciences*, Porlamar, Venezuela., pages 389–394.

#### 1.4.5 Haptic Rendering using Tangent-space Atlases

Chapter 6 details a fast global approach for haptic rendering relying on image-based Hybrid Rugosity Mesostructure Atlas in tangent space, allowing real-time accurate perception of surface relief detail in large models meshes, validated in numerical, statistical and perceptual metrics.

The approach builds an atlas of *per face* displacement textures and corresponding normalmaps in substitution of actual geometry, which grows as a generalization of the rugosity

mesostructures for accelerating haptic rendering described in Chapter 4.

A two-step global procedure encodes all relief detail by generating an image-based Hybrid Rugosity Mesostructure Atlas (HyRMA) shell, capturing the surface detail of a dense mesh by obtaining depth differences in piecewise triangular prism volumes extruded from the triangles of the simplified version of the same mesh, and then warping all depth information into corresponding orthogonal triangular prisms. The warped depth differences (and corresponding *unwarped* normals) are then stored as a non-contiguous 4-channel (RGBA) texture atlas in tangent space. In the rendering step, the 4-channel tangent-based texture is then used as input to a relief shader which is applied to a simpler mesh, constructed as a highly decimated versions of the original denser mesh, thus rendering the original mesostructure detail with a lot less triangles.

The first contribution of this global approach is guaranteed geometric continuity [at least of order  $G^0$  and  $G^1$ ] across all faces. The second one is producing accurate correspondence between surface detail visualization and the perception of its fine features without compromising rendering framerates, accounting for a qualified loss of minor mesostructure detail.

The method is validated threefold: *geometrically*, by sampling and verifying a trajectory, showing no surface differences between pure geometric and HyRMA-rebuilt models; *statistically*, by obtaining haptic atlas rendering metrics showing optimal haptic/visual rendering framerates; and *perceptually*, by user testing and usability controlled trials measuring accurate haptic sensation of large meshes' fine features at interactive rendering rates.

#### 1.4.5.1 Contributed publications from this area

THEOKTISTO, V., FAIRÉN, M., AND NVAZO, I. (2013). Generalized haptic relief atlas for rendering surface detail. In *GRAPP & IVAPP 2013: Proceedings of the International Conference on Computer Graphics Theory & Applications and International Conference on Information Visualization Theory & Applications*, SciTePress, Editors: Coquillart, S., Andújar, C., Laramée, R. S., Kerren, A., and Braz, J. Barcelona, Spain, 21-24 Febr., 2013, pp 191–196.

THEOKTISTO, V., FAIRÉN, M., AND NVAZO, I. (2014). Tangent-space mesostructure atlases for accurate real-time haptic rendering. *Presented in the XXIV Spanish Conference in Computer Graphics (CEIG'2014)*, University of Zaragoza, Spain.

THEOKTISTO, V., FAIRÉN, M., AND NVAZO, I. (2015). Tangent-space mesostructure atlases for accurate real-time haptic rendering. *Article been reviewed for publication in a refereed journal, 12 pages.*

### **1.4.6 Discussion**

Lastly in Chapter 7, appropriate conclusions from this work are drawn, its principal achievement being that the techniques described herein constitute a quantifiable better alternative for haptic rendering, pointing to future work in the area with some intended extensions for real-time synthesis of multiresolution haptic textures.





## *Related work*



*If you want to make an apple pie from scratch, you must first create the universe.*

*Carl Sagan*

**H**aptic rendering depends on “as fast as possible” real-time interactions at feedback rates around 1 KHz. Avoiding slowdowns and/or interruptions requires devising strategies that attack those areas where suboptimal sampling might impede timely manipulation and response. Haptic interaction performance is affected by (i) the [underlying] Collaborative virtual Environment’s distributed/threaded architecture, (ii) Collision Detection Computation Strategies, (iii) Haptic Rendering algorithms, and (iv) Image-based Haptic Approaches for Geometry Substitution. In the following sections we will address separately relevant research on each of these topics.

### **2.1 Collaborative Virtual Environments (CVEs)**

Distributed environments have been around since the introduction of the first networks. Scope and complexity have kept pace with distributed systems evolution, migrating towards distributed processing, data sharing, multiple execution threads and sophisticated display technology.

Virtual Reality (VR) and Augmented Reality (AR) tools have been applied in all engineering fields in order to avoid the use of physical prototypes, to train in high risk situations, and to interpret real or simulated results. In medical applications they help patient monitoring, interpretation of scanned data and surgery planning. In architectural settings enable designing, building, visiting and stress-testing upcoming facilities. In these Virtual Reality Environments or VREs, individual users inspect 3D scenes, navigate inside models and manipulate objects and properties.

However, most implementations of VREs usually begin as standalone applications, with collaboration requests arising from the natural desire of exchanging experiences. Allowing several clients to collaborate on the inspection of a model usually requires the development of a whole new application with distributed capabilities, adding network communications, and in general confronting code portability problems due to the absence of a migration strategy.

Computer Support for Cooperative Work(CSCW) as defined by Schuckman *et al* [119] is an umbrella term for distributed applications in which multiple users collaborate toward common goals, under a high level event notification and message passing architecture. When combined with several degrees of information sharing, 3D data visualization and real world user-interaction metaphors they become Collaborative Virtual Reality Environments or CVRE's. Remote participants using visual identities (called *avatars*) may navigate inside the virtual space, interact with other remote avatars, and propagate changes to neighboring objects.

### 2.1.1 Copresence as shared sensations

A taxonomy of the meanings and subtypes of copresence structuring human interaction in virtual environments is proposed by Zhao in [160], defined as the relationship between the physical conditions and the sense of being with others and how the former conditions the latter. Masa [79] describes the sensations users need to perceive for a complete experience in a CVRE:

- *A shared sense of location in (3D) space.*
- *A shared sense of (real) time.*
- *A shared sense of co-presence (using avatars).*
- *A shared (external) communication channel (text, speech or video).*
- *A sharing mechanism for object manipulation.*

Any incarnation of a CVRE aiming to produce realistic interactions with users must address and implement solutions for all five of these expected sensations.

### 2.1.2 Copresence CVE implementations

Among the first available distributed virtual environments was DIVE [14], in which all users were equal peers, communicating among themselves to synchronize state, environment and display information. In MASSIVE-3 [44], another early DVE, the approach used is the transmission of images and video in Virtual Reality (VR) or Augmented Reality (AR) environments. A review of simulation CVE's developed at the US Air Force Institute, such as SIMNET and its successors is presented in [124], addressing issues in human-computer interaction, virtual reality, software engineering, 3D graphics, scene modeling/object modeling, and artificial intelligence. Duce *et al.* [31] characterize reference models for CVEs and

the impact they have on the degree of collaboration, based on the increase of complexity in the collaboration. A different approach is presented in [76], where the CVE is included into a library called REPO-3D, which allows the migration/replication of graphic objects over the network.

Different network technologies can be used to enable distribution on a CVE (RPC, BSD-sockets, Java RMI, DCOM, CORBA, etc.). AVOCADO (later AVANGO) [145], DIVE, SIMNET, NAVL [151], NPSNET-V (Java-based) [13], and Distributed Open Inventor [49] apply different solutions, using multiple execution threads where each one has an image of the other participants in the interaction. COVISE [107], an OpenInventor simulation implementation, uses a request broker to handle client connections, and a one master–several slaves approach using `exec` calls for local processes and `rexec/rlogin/rsh` for processes on remote computers. Audio channels and the use of 3D markers help steer the collaboration. In Distributed Open Inventor (DOI), each client has a copy of the global scene graph and the synchronization is done by using *rsgp* (*replicated scene graph protocol*). Some local variations on the graph are allowed, such as different levels of detail, “ghost” objects and local changes of color or texture.

Each client requires, thus, at least a partial image of the scene graph. Zeleznik *et al.* [158] use the scene graph as a communications bus instead of a tree, whose nodes are sited at different network nodes and are accessed by synchronized access mechanisms by heterogeneous applications. Diverse [61] uses remote shared memory and UDP network datagrams for a rapid memory interchange. Although this protocol does not guaranty the reception of messages, the quick reception of almost complete information allows to maintain the coherence among participants.

The earlier treatment of arising temporal inconsistencies in distributed systems due to network delays was initially described by Lamport [66], while using logical clocks to resolve temporal causality relations between events [108], and their contribution to reduce shared scene corruption are detailed in [80] and the VOODIE system [81]. Greenhalg [43] describes a technique for embedding temporal links in the Massive-3 CVRE, which allows recursive self-reference for models, such as showing a small version of the model (a maquette) as a 3D-map within itself.

The most sophisticated approaches delegate clients’ network management to components outside of the CVE. Some of them use customized solutions, like Octopus and Tweak for VR-Juggler [48] or CAVERNSoft for CAVELib [70]. Other are based on the use of the CORBA standard [27] for distributed services over the network, having a central object registry and a localization service. CORBA-based solutions need to implement an Object Request Broker (ORB) to resolve requests for object references, using an Interface Definition

Language (IDL) to publish the interfaces of objects in a language-independent manner. This allows the development and integration of heterogeneous systems with adaptive services related to network performance, useful to insure QoS on high traffic environments, but this approach may be too complex and heavy for an expected short number of similarly configured clients.

Blue-c [46] combines live video feeds in a distributed scene graph based on the OpenGL Performer toolkit and an API that minimizes synchronization overhead among local and shared objects, under the CORBA-based middleware. The NOMAD framework [155] additionally implements some real-time CORBA extensions to deliver environment state changes to a number of clients in a timely manner. Hubbold *et al.* [56] in the GNU/Maverik/Deva VR system take a different approach, using a modular micro-kernel architecture with default callbacks and immediate-mode rendering, having the external Deva module in charge of the collaboration services. Yet another option is to tinker at a lower level with a custom communications protocol such as *vrtp* (*virtual reality transfer protocol*) [11], implemented by a plugin architecture under the Bamboo API [150] for networked VE's.

Uneven network speeds and workstation performance are a hindrance for truly massive CVRE's. A solution currently used in P2P environments [16] establishes the *SplitStream* or *bit torrent* content distribution system. Objects are split in segments and load is distributed among all the participants, so the more clients there are, the faster distribution works. For truly massive architectures, VELVET [26] introduces an adaptive hybrid architecture through an adaptive filtering scheme based on multicasting that allows users in a supercomputer with high-speed networking to successfully collaborate with others in not-so-powerful systems or under a slow connection.

### 2.1.3 Massively Multiplayer Online Role Playing game engines

Most recent development of large scale CVE implementations come from widely deployed online games, or Massively Multiplayer Online Role Playing Games (MMORPGs), whose huge numbers of client users really put a strain on both client-server and P2P models. Participants enter and leave continuously, which in turn creates large imbalances in P2P networks. Hu and Chen [55] solve the neighbor discovery problem proposing a Voronoi-based Overlay Network (VON), a simple and efficient design that maintains fully-distributed P2P topologies in a low-latency and message-passing efficient manner. The requirements and alternatives of P2P communications engine for massively multiplayer online game are analyzed by Krause [65] and Schiele *et al* [118], while in Strassburger, Schiele and Becker [125] peer-to-peer techniques for MMORPG functionality are used to distribute and balance loads at the cloud server centers .

The largest installed user base of any MMORPG is by far WorldOfWarcraft® (WoW), developed using a P2P networked game engine, which we will refer by the acronym WoWge. Svoboda *et al* [128] and Suznjevic *et al* [127] have analyzed the regular game traffic to improve P2P performance, which lags as the number of users increase. The other hugely popular game is the Quake® family of First Person Shooter (FPS) games [100], based on the MPPORPG engine of another FPS game, Doom 3®. It uses a multi-server architecture using a Real-Time Framework (RTF) and a state replication approach that allow scaling up to large numbers of “rooms” with many users each.

Miller [83], using data gathered from real sessions of WoW, reinterprets the traffic in a client-server model and as a result determines that P2P paradigm is not suitable for huge number of users. Alternate workload models are explored and preliminary conclusions presented. Using realistic workloads it is shown that a fully decentralized DVE cannot be deployed to today’s consumers, regardless of its overhead, and that a multiserver client-server model has the lowest latency times overall.

In general, recent games are developed using either private engines or increasingly using available Networked Game Software Development Kits, such as the Unreal Engine [58] and Unity3D [133], which are architecture-agnostic and allow implementations of any communication protocols. Dionisio, Burns & Gilbert extend previous results and surveys, concluding that as network speed increases, most synchronization and latency problems will arise from increased realism in media, and less for game state changes, allowing for a scalable hierarchy of multilevel servers.

More rounded and up-to-date surveys treating all issues involved can be found in Yahyavi and Kemme [157], and in Buyukkaya, Abdallah & Simon [12], presenting a comprehensive overview of current strategies, techniques and peer-to-peer solutions for massively multiplayer games, their advantages and drawbacks.

## 2.2 Collision detection computation strategies

Basically, a collision is the result of a spatial query asking whether two geometric objects intersect at some point in time. Objects may be triangle meshes, NURBS surfaces or implicit algebraic expressions. It is mainstay in many applications such as videogames, physical simulations, computer animation, robotics, and structural engineering. A review of collision detection techniques and data structures are described in [59]. Strategies are categorized as geometric or algebraic, addressing *spatial/temporal intersection*, *trajectory parameterization*, and *object partition representation*, either as rigid body collisions (undeformable objects) [88] (highly used in videogame development and haptic manipulation [93, 141]) or

soft body collisions (deformable objects) [23].

The most basic intersection is the ray-triangle intersection, needed both for ray-tracing and collision detection, as implemented in Möller's and Trumbore optimized CPU approach for Ray-Triangle [87], and Triangle-Triangle intersection [86].

Most techniques avoid exhaustive detection by enclosing or partitioning objects into fast-to-discard simpler convex *Bounding Volume Hierarchies* (BVH's) [33]: Axis-Aligned Bounding Boxes (AABB), Rectangular Swept Spheres (RSS), Oriented Bounding Boxes (OBB) [42], and Convex Hulls. A hierarchy of inclusive bounding volumes is built to discard quickly large object sections, such as a linear bounding volume hierarchy (LBVH), kd-trees, BSP (binary space-partitioning) and others implemented as trees of bounding volumes [159]. There exist other BVH schemes, such as OBBTree [41], BOXTREE [6] and BSP tree [33].

### 2.2.1 GPU general computing

Graphics rendering has been migrating towards a programmable model of vertex and pixel (fragment) shaders [74] in current graphics cards. Pre and post-processing tasks that once were done on the CPUs, such as bump mapping, depth of field, shadow mapping, dynamic texturing are now implemented in the GPU, allowing real-time rendering of complex effects [90].

The SIMD paradigm [32] provides the GPU's considerable multicore power (from 128 to 1024 cores *now*) for the stream processing of millions of vertices and pixels per second, using separate execution threads. General-Purpose Computation on Graphics Hardware [97] harnesses this parallel computation power by storing generic data into vertex and texture memory, and using vertex and pixel shader programs to compute numerical calculations (in matrix) form, instead of images. Its applications range from model construction [67], accelerated 2D image processing [99], fast collision detection [63], and others. A complete up-to-date survey of current applications is to be found at [68].

### 2.2.2 GPU-assisted parallel intersection computation

General-Purpose Computation on Graphics Hardware [97] harnesses programmable graphics processors to solve vastly complex problems, sending data as texture memory to shader programs for some number crunching instead of image rendering. Georgii, Krüger, and Westermann [39] propose a collision model based on intersecting mesh polygons with no pre-filtering object hierarchies. At the rendering step, geometric GPU shaders pack potential collision info into textures for detecting the collision in the later rendering phase. It reports nice interactive rates and reference collision times for several test meshes, both

rigid and deformable. Pabst *et al* [98] use a hybrid multi CPU-GPU approach obtaining also good collision rates in the milliseconds range.

Nykl, Mourning, and Chelberg [92] present a technique for interactively deforming and colliding mesostructures at a per-textel level, able to reduce traditional 3D geometrical deformations (vertex-based) to 2D image space operations (pixel-based) that are parallelized on a GPU. It requires no preprocessing time and storage requirements of one additional texture or less. More recently Shumskiy [121] realizes a comparative study on ray-triangle intersection algorithms in GPU, giving reference times for Möller's and others intersection algorithms. However, the last three approaches use some form of geometry prefiltering (BVHs and others) throughout, so their rendering times are not fine grained enough to extract the time spent in pure collision detection. The collided models are in the range of a hundred thousand polygons, although for some tests they does not report the polygon count of the meshes involved, just the relative ray-triangle intersection times.

Up to 2007, all GPU computations consisted in ingenious and oftentimes tortuous harnessing of the vertex-fragment programming framework for parallel computations. In that year appears NVIDIA's Compute Unified Device Architecture (CUDA™) API/SDK [21, 116], providing a SIMD parallel programming framework with concurrent threads and simultaneous kernel code execution at GPU streaming processors, based a higher API in a C-like language for general GPU programming.

The fundamentals of CUDA programming are:

- Kernels: code executing in parallel at each GPU core.
- Threads: separate execution environments for kernels
- Blocks: collections of up to 512 threads organized as unidimensional, bidimensional or tridimensional constructions.
- Grids: a set of blocks organized in unidimensional or bidimensional o fashion.
- Memory: may be Register, Shared (for all threads of a block), Global (for all blocks), Texture (uses filtering) and Constant (read-only).

In CUDA, when an application requests execution in parallel, a grid is configured as a mix of sequential and parallel blocks. Each block sets up its own threads and kernels. A single GPU card has several Streaming Multiprocessors or SMs, each capable of executing up to 8 blocks or *CUDA cores* simultaneously. Each SM administers groups of 32 threads at the same time. There is no communication among blocks, and they may execute in any order.

Given the parallel nature of collision detection schemes, especially those needed for haptic collision/interaction, CUDA-based implementations are straightforward, and quite fast for this type of problems.



## 2.3 Haptic rendering

The term *haptic rendering* is initially defined by Zilles and Salisbury [161] as the real-time generation of feedback force responses according to users interactions with virtual objects, by computing collisions against a force-feedback device placed in a 3D environment. A high priority event loop checks for collisions against the geometry, after which it generates at the device new forces and torques of varying direction and magnitude [78].

An early effort to measure haptic discrimination of 2D textures was the Sandpaper System by Minsky and Lederman [84], where users manipulate a force-feedback joystick to traverse simple textures and evaluate qualitative roughness differences. Siira and Pai [122] incorporated a stochastic model of physically correct surface properties to produce appropriate textural feel, including friction and lateral forces. Costa and Cutkosky [22] generated fractal rugosities on flat surfaces and measured perception thresholds.

When used as an aid for navigating a space, its short range reach requires space exploration strategies, such as a moving bubble for navigation [28], a workspace drift control allowing perception discrepancies between visual and haptic space [20], or a force-filled constraining movement [156]. Collaboration across networks allows simulation of real-time activities such as stretcher-carrying [57], but it brings its own set of latency and simultaneity problems that may cause oscillations in the interaction.

With device sampling rates already in the 1000 Hz range, (the PHANTOM™ Omni and the HAPTICMaster™ device [35]), efficient haptic-interaction techniques may go beyond the simple detection of geometric primitives, towards allowing real-time rendering of arbitrary surfaces of irregular detail, conveying spatial and material properties. All this without forgetting its other role as an user-interaction device for high level event acquisition, recognition of tactile “icons”, and general haptic user interfaces or HUIs [77].

A first approximation for haptic rendering among objects using a texture atlas is described by [96]. Object intersections are tested first by collisions between low resolution meshes of the objects and a height function of a surface patch is approximated by a composite mapping function. Collision and haptic modules run in its own threads, and synced visual rendering is accomplished at a dedicated *separate* computer, to avoid impacting the haptic update rate.

Using a third object as an extended probe allows real-time texture differentiation and shape perception [95]. Detecting friction among objects is achieved by rubbing simulated known material against each other [109], calculating friction forces using common physical models. Fontana *et al.* [36] analyze the mechanical design and software computational issues arising when kinaesthetic devices have to be integrated for model perception.

These efforts choose among several alternatives for modelling and rendering surfaces. Gregory *et al.*'s H-Collide [45] uses hybrid hierarchical representation of uniform grids and trees of tight-fitting oriented bounding boxes, whereas Johnson [60] uses a pure geometric render approach for arbitrary polygons using neighborhood proximities. Morgenbesser and Srinivasan in [89] proposed the method of *force shading*, akin to *Phong shading* and *bump-mapping*. The *force response* vector is interpolated from nearby vertices, but it is unable to elicit accurate geometric up-down perception. A global procedure for mapping a gray-scale image as a displacement map for point-based haptic rendering is given by Ho *et al.* [53]. It works only for convex objects of genus 0, without any assessment of perceived sensation fidelity.

Inadequate modelling or suboptimal rendering produce instabilities in the force response, as shown in the work of Choi and Tan [17–19]. Collisions are detected against a coarse geometry and then against a second micro-geometry layer. Incorrect renderings when traversing concave foldings are identified but not addressed. A similar approach for geometric sculpting with 2D textures and a haptic stylus is used by Kim *et al.* [62]. Potter *et al.* [103] provide a simple model to perceive haptic variation in large heightfield terrains, detecting collisions against the terrain's dataset patches.

Sreeni *et al.* [123] describe the problems of haptic rendering of objects at different scales for cultural heritage applications. For pure geometric-based haptic rendering relying on direct probe-polygon collisions, Melero *et al.* [82] detail a collision algorithm, building an indexed Bounding Planes Octree data structure (BP-Octree), used for haptically rendering polygonal models of upto one million polygons at acceptable interactive query rates.

An extended survey of all haptic rendering techniques can be found in the work of Laycock and Day [69]. From the latter review, it follows that haptic rendering approaches have relied either on straightforward collisions against the mesh's triangles or a NURBS parameterization of the same. Also, in a survey by Varalakshmi *et al* [24] shows the absence of formal treatment regarding the use of heightfield displacements for haptic rendering, a lack of unified testing frameworks for measuring quantitative and qualitative differences among rendering approaches, and no usability testbeds for standard models and surfaces.

Finally, in the more recent survey by Weller [153], there are considerations for implementing the detection of haptic spatio/temporal interactions using bounding volumes and packing strategies, describing collision detection as the interplay of force and torque of geometric volumes.

### 2.3.1 Taxonomies for haptic interaction

Haptic technologies can play multiple roles due to the multimodal nature of its reinforcing interaction technique. Deciding which one is more appropriate for the data at hand requires a topical classification of haptic capabilities. A preliminary classification of the distinct styles of using haptic perception for purposeful activity is reported by Kirkpatrick and Douglas in [64]. It characterizes haptic interactions according to several categories to structure a unified perspective on haptics use: Haptic Hardware, Haptic Perception, Haptic Detail Modeling, Haptic Rendering Algorithms, Haptic Space Navigation, Haptic User Interfaces and Haptic Multiuser Collaboration.

#### 2.3.1.1 Haptic hardware

Haptic hardware is varied, and many involve a force-feedback loop (which may become unstable) repeatedly sampling the force-response of a special probe. May be based on

- *One point sample*, allowing for general force sampling.
- *Two or more points simultaneous sample*, allowing torque.
- *Contact surface*, which senses shape and other properties.

#### 2.3.1.2 Haptic detail modeling

The approach used to model objects' surfaces or volumes influences the render algorithm used. The models may be based on:

- *Geometry*, rendering the surface as detailed polygonal meshes.
- *Surface Patches*, which may or not be rendered geometrically.
- *Surface Relief Detail*, in which a haptic texture is sampled in lieu of the actual surface. On its own, haptic textures may be
  - Geometry-based, (polygon meshes, NURBS, point clouds).
  - Texture-based (bump maps, force fields, height fields).

These approaches may allow managing LOD in the visual and haptic resolutions, and a repository of relief texture.

### 2.3.1.3 Haptic Rendering Algorithms

To relate the haptic probes with the models, there are several rendering algorithms that may be used individually or together to render the quality of contact with the surface:

- *Pseudo-haptics*, simulating texture with pointing devices such as mice or tablets by varying the speed and direction of visual displacement.
- *Collision detection*, which may require storing objects' surface in some fast searching hierarchical structure, such as octrees. When the probe hits the surface, there is a feedback response.
- *Force-field composition*, requires modulating the probe response according to a spatial force model able to place a dynamic 3D force field that takes into account the operator's exerted force.
- *Antialiasing (reduction)* of haptic artifacts arising from model or algorithm discretization, especially when dealing with LOD.

### 2.3.1.4 Haptic space navigation

In virtual environments that allow navigation and inspection, “reachability” in haptic rendering is a term akin to “visibility” in graphics rendering. However, the “horizon” of a haptic probe is much smaller than the visual horizon. On the other hand, an optimal visual rendering rate is a comfortable 60Hz when compared to the high sampling rate of 1000Hz of current haptic devices. So the strategies are more or less the same:

- Complete haptic space representation.
- Local caching (with or without prediction).
- Moving cell (or “bubble”) (with or without prediction).

### 2.3.1.5 Haptic multiuser collaboration

Multiuser interaction with haptic devices suffers all the ailments of distributed applications, such as uneven traffic, network latencies and loss of connection, amplified by its high speed sampling rate. A noticeable lapse between force emission and force response may produce wild and sometimes dangerous oscillations in the devices. Thus it requires

- Multiresolution broadcasting model, minimizing sent packets (up to a point).
- Treatment of latency, either allowing simultaneity or a buffered response.

### 2.3.1.6 Haptic Perception

From the operator's side, he has an objective in mind, which is to perceive scene and object properties. These may be grouped as

- *Tactile*, texture or 2D form: (hard/soft, smooth/irregular, flat/slope, slide/hit).
- *Kinesthetic*, absolute and relative spatial awareness: static (3D shape, length, weight); dynamic (strength, viscosity, compressibility, torque, vibration).

### 2.3.1.7 Haptic user interfaces

Haptic devices may act as enhanced pointing devices, in which the point-and-click metaphor allows more modes and events than in typical pointing devices. Haptics is very limited as stand-alone input method, and needs an assisting technology, such as auditive or visual correspondence, to avoid disorientation. A Haptic User Interface (HUI) metaphor may define the following high level haptic events:

- *selectors*, possible position/orientation change.
  - hit, touch/untouch, slide/bump, grasp/release, hold/feel/weigh.
- *mutators*, modifying object geometry and/or surface properties).
  - cut/join, press/stretch, break/fuse, smooth/sculpt).

## 2.3.2 Haptic performance metrics

Haptic Rendering perception must be evaluated by devising testable experiments and appropriate measurement procedures. Guttman [47] states which appropriate measurement protocols must be followed in a general experimental setting to obtain sound statistical inferences, and also how to avoid wrong experimental setups and common formulation pitfalls. Otaduy and Lin [94] review the physical and psychophysical experimental methodology applied at the whole haptic rendering pipeline, including parallel computation and surface patch heightfield interactions. More recently, Samur [114] presents a categorization of haptic interaction tasks, performance metrics and experimental measuring tests as a first step towards a standardized evaluation method of tactile and force feedback devices.

## 2.4 Image-based haptic approaches for geometry substitution

The techniques presented in Chapter 6 rely on specific space deformations of volume meshes to produce warped pairs of relief and normal textures. Then the textures are unwarped and sampled at a later time to render the original relief detail. The following relevant milestones are pertinent in understanding the context of that part of our work, going from spatial deformations to appropriate texture atlas representation.

### 2.4.1 Deformation and warping

Gain and James [38] present a good survey for spatial deformation methods in general, in which they describe a family of modelling and animation techniques for indirectly reshaping objects, interactively warping the surrounding space and applicable to a variety of object representations. Deformations are controlled by tools of varying dimension –points, curves, surfaces and volumes– and classified by user-centered criteria of versatility, ease of use, and efficiency and correctness.

### 2.4.2 Texture atlases approaches

As far as shown in the relevant literature, there has been few insights of the problems surrounding the use of heightfield displacements for image-based rendering, such as concave areas and holes, the emphasis being made in texture stitching and smoothing. Levy *et al.* [73] describes the Texture Atlas method of mapping any surface in an optimal one-to-one correspondence of texture and geometry along contour lines..

Carr [15] describes a one-to-one mapping from an object's surface into its texture space. The method uses the graphics hardware to rasterize texture coordinates and colors directly into the atlas. Levy *et al.* [73] describe the Texture Atlas multi-chart method of mapping any surface in an optimal one-to-one correspondence of texture and geometry along contours. Sander *et al.* [115] map the surface piecewise onto charts of arbitrary shape with a zippering algorithm that creates watertight surfaces with reduced parametrization distortion. Robust methods to achieve consistent mappings of surfaces of arbitrary genus are described in Praun *et al.* [104] and Rossi and Bergamasco [113].

Purnomo *et al.* [105] solve the problem of seamless texture stitching across boundaries, but the approach has been used only for texture coloration, and so far it has not been used to represent geometry differences as texture displacements, either in visual or haptic ren-

dering. Gonzalez and Patow [40] solve spatial discontinuities of multi-chart parameterizations by a bidirectional mapping between areas outside and inside the charts, and stitching triangles by linear interpolation between non-adjacent texel values, done at low computational cost and with small memory footprint.

### 2.4.3 Pixel-based relief rendering

Displacement mapping approaches subdivide the original geometry into a larger number of triangles which are displaced according to a 2D height map. Although geometrically accurate, they suffer for poor interactive performance.

A complete survey on displacement mapping approaches can be found in Szirmay-Kalos and Umenhoffer [129]. A common characteristic of the approaches about to be described is that they rely on iteration within modern shader implementations (Cg, GLSL, DirectX9), since in one way or another they implement ray casting, per-pixel interception, impostor representation with multisampling of depth relief and normal textures, and correct silhouette calculations for rendering geometric detail.

Hirche *et al.* [52] devise sampling a displacement map per pixel in a shader. Triangles of the base mesh are extruded along the respective normal directions and then the resulting prisms are rendered by casting intersecting rays with the displaced surface. Polcarpo, Oliveira & Comba [102] use a purely image-based approach using two (front and back) depth textures for real-time relief mapping on arbitrary polygonal surfaces. Baboud, Lionel & Décoret [5] extend the former approach rendering geometry by using up to six relief (displaced-mapped) impostors, corresponding to the faces of the object's bounding cuboid, describing a fast shading of geometric objects using either a two-sided back/front map or a six-sided cube map.

In Nießner and Loop [91] is introduced a smooth analytic displacement function, stored in a GPU-friendly tile based multi-resolution mip texture format. It computes per vertex adaptive tessellation factors and select the appropriate pre-filtered mip levels of the displacement function, that does not need a pre-computed normal map.

Parallax Occlusion Mapping (POM), described by Tatarchuk [131, 132], is a simpler procedure based on a linear search in texture space, shifting the texture coordinates along the view ray direction according to depth values, and skipping self-occluded pixels in the current view direction. It is expanded by Dasbacher & Tatarchuk [25] by embedding the displaced surface in a triangular prism volume grown from three slabs along the vertices' normals. Ray marching texture gradients (heights) are computed per tetrahedron (three (3) tetrahedrons to a prism).

A successful approach for relief mapping with correct silhouettes uses a fragment shader

based on *relaxed cone step mapping*, as implemented by [101], using an additional pre-computed pixel “cone map” for mesostructure generation. Cones represent empty space between relief crests corresponding to pixels in the texture that can be safely skipped in the ray-casting process, thus accelerating geometry generation.

Santos *et al.* [117] describe solid height-map sets, capable of representing overhangs or self-folding surfaces and occluding objects, together with a fast visualization algorithm with performance independent of the original mesh size.

A different approach is described in Timonen and Westerholm [144] that also takes into consideration self-occlusion and self-shadowing, using a parallel CUDA implementation which is only applicable for planar heightfields. Visibility for each point in a heightfield is determined as the exact horizon for a set of azimuthal directions in time linear in height field size and the number of directions. Surface is shaded using the horizon information and a high-resolution light environment, producing detailed extended shadows.

In conclusion, several parallel image-based techniques have been devised to represent geometry as texture for visual rendering, and the hypothesis is to see whether the same approaches are valid for geometry substitution in image-based haptic rendering.





# *Embedding Collaboration for Virtual Reality*



*In life, unlike chess, the game  
continues after checkmate.*

*Isaac Asimov*

**V**irtual Reality (VR) and Augmented Reality (AR) tools have been applied in all engineering fields in order to avoid the use of physical prototypes, to train in high risk situations, and to interpret real or simulated results. In medical applications they help patient monitoring, interpretation of scanned data and surgery planning. In architectural settings enable designing, building, visiting and stress-testing upcoming facilities. In these virtual reality environments or VREs, individual users inspect 3D scenes, navigate inside models and manipulate objects and properties.

A CVE (Collaborative Virtual Environment) uses local and remote network nodes to build a unified description of a virtual shared space, allowing the interaction among autonomous and human-controlled entities. This interaction may have educational, research, planning, design, evaluation, training, and/or simulation purposes. The shared space illusion depends heavily on the timely and lossless transmission of all interaction packets, and may become inconsistent if participants fall out-of-sync because of increasing network latency rates.

A Collaborative Virtual Reality Environment (CVRE) is a CVE representing a simulated 3D universe, in which one or more users take visual identities inside the environment known as avatars. Avatars can navigate around the 3D world, be aware of and collaborate with other avatars in real time, and synchronously propagate changes in the objects and the environment. Environment and object manipulation require sophisticated interaction models and user interfaces.

However, most implementations of VREs usually begin as standalone applications, with collaboration requests arising from the desire of exchanging experiences. Allowing several clients to collaborate on the inspection of a model usually requires the development of a whole new application with distributed capabilities, adding network communications, and in general confronting code portability problems due to the absence of a migration strategy.

### 3.1 Collaborative Virtual Reality Environments (CVREs)

We have developed a taxonomy of the meanings and subtypes of copresence that structure human interaction in virtual environments, extending those proposed by Zhao in [160], defined as the relationship between the physical conditions and the sense of being with others, and how the former conditions the latter.

The design of collaborative virtual environments must choose to implement some alternative of compliance with the following orthogonal requirements:

- i) *session awareness*
- ii) *scalable topology*
- iii) *network transmission*
- iv) *collaborative user interaction features*
- v) *object complexity*

The first three apply to generic CVEs and the last two are specific to CRVEs.

#### 3.1.1 Taxonomy of collaboration in virtual reality environments

The requirements presented below provide the designer with a recipe for creating a compelling sense of co-presence in virtual environments, no matter what the feature set is.

- i) **Session awareness:** Persistence, the temporal or permanent effect of user interactions have in the CVRE system [71]; may be described as:
  - *Participatory:* The CVRE exists only while the participants are in it, and shuts down when all participants leave the environment (WOWge, Quake).
  - *Journalled:* Session scripted for later state recovery, allowing recording and re-playing of 3D temporal annotations to guide other clients (Massive-3).
  - *Continuous:* The CVRE is always active. A simulation make change scene and objects even if no clients are connected (SIMNET).
- ii) **Scalable topology:** Scene sharing schemes among participants [31] can range from:
  - *Homogeneous replication using broadcast*  
Each client maintains a complete replica of the shared environment. Messages across the network maintain state information. No central control; a new client

has to wait some time to get information sent by other clients broadcasting changes (SIMNET, DIVE).

- *Shared-centralized on a server*

Classical client/server model one with one scene being shared by all, residing at a central server (CAVERN, NPSNET-V). When the server fails it brings down all the clients.

- *Shared-distributed with client/server groups*

Several groups of servers and clients. Uses same scheme as mobile phones cells, in which clients are connected to the nearest or least busy server (DIVE, Massive-3, Octopus, NOMAD, VELVET, Quake).

- *Shared-distributed using P2P actualization*

Peer-to-peer connections among all participants, either directly or using a third party relay (broker). Changes are atomically broadcasted to all participants. It comes in two flavors:

**P2Pr:** Replicating the same scene graph at each node (DOI, COVISE), with objects stored locally. Synchronization is done by callbacks managing of session coherence.

**P2Ps:** Sharing objects across the network in a distributed scene graph with remote objects (Diverse, Blue-c, GNU/Maverik, WOWge).

- iii) **Network transmission:** Using an appropriate protocol to the expected message flow, such as choosing UDP or TCP/IP packets; use of broadcast in a LAN (as in SIMNET), unicast (one-to-one, all) or multicast (one-to-many) addresses (CAVERN, DIVE, Diverse, DOI, GNU/Maverik, Massive-3, NOMAD, NPSNET-V, Octopus, VELVET, WOWge and Quake); procuring reliability, bandwidth and minimizing network latency.
- iv) **Collaborative user interaction features:** the collaborative set of desirable manipulation and visualization interfaces, teleconference capabilities (chat, video and audio), flexible support for model construction, synchronous and asynchronous collaboration modes, adaptive multi-resolution strategies, interoperability standards, and virtual space shared utilization. Crucial features for CRVEs are: 3D annotation and action indicators for remote event notification, multiple alternate views, selectable avatars, and expected low latency response times.
- v) **Object complexity:** determines the network broadcasting cost of object and scene changes [11], including LoD and multi-resolution models, as:

- *Light objects*, Short messages containing event state and control information, requiring low latency, high-speed networks, such as trackers, sensors, status information) (All systems).
- *Remote references*, local network references shadowing remote objects (All but SIMNET).
- *Heavy objects*, Big objects requiring reliable transmissions, but small enough to reside in local memory, (e.g. object 3D geometry, avatars or cameras) (All systems).
- *Real-time streams*, large-segmented data. Data so big it has to be transmitted in pieces and/or continuously, (e.g. big geometric objects, volume information, textures, video, audio, etc.). (CAVERN, Blue-c, Massive-3, WOWge, Quake).

From the above, it is evident that many CVREs use multicast addresses for UDP or TCP/IP communications. The most recent ones lean towards P2P or small client/server topologies, with replicated or shared scene graphs. Only CAVERN, Blue-c, Massive-3, WOWge and Quake integrate segmented data such as video or audio feeds, while some of the others resort to variable resolution schemes or out-of-core segmentation.

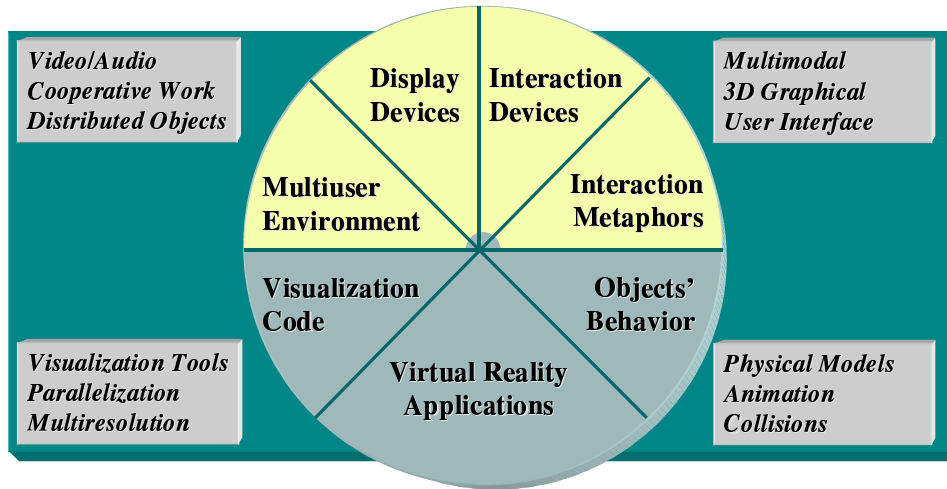
Massive-3 is the lone provider of journaling mechanisms for interaction recovery. Use of *Avatars* for self-representation are a common feature, but none allow multiple perspectives. Only DIVE, Massive-3, VELVET, GNU/Maverik, WOWge and Quake seem capable of handling large user loads or huge data models.

As far as the former reviews show, there is no clear strategy allowing an orderly and easy migration path from standalone VR applications to collaborative ones.

### 3.1.2 Characterization of collaboration in virtual reality environments

Treatment of remote collaboration capabilities in a CVRE can be characterized by the categories summarized in Table 3.1. The relevant references can be found in [136]. Figure 3.1 details all major software components present in a complete CVRE, requiring sophisticated user interaction models for domain and object manipulation. The categories highlighted in Table 3.1 are those that allow the designer to specify the most suitable feature set for creating a collaboration framework and a sense of presence in virtual environments. The first three apply to generic CVEs and the last two are specific to CVREs:

- i) *network transmission*: decide which network architecture is best suited to the expected message flow in the environment;



**Figure 3.1** Components of Collaborative Virtual Reality Environments. The items placed at the upper half of the circle are those directly related to collaboration issues.

- ii) *scalable topology*: choose a scheme for information sharing and communications among several participants;
- iii) *object complexity*: measure the network performance cost of broadcasting object changes [11];
- iv) *environment persistence*: decide the temporal or permanent effects of user interactions within the environment have in the CVRE system [71]; and
- v) *user interaction* include those user interface features desirable for a CVRE.

All studied CVREs were profiled with these categorization, shown in Table 3.1, as part of this research, and the global profile can be seen on Table 3.2. An example feature implementation based on these tables is described in subsection 3.4.1.

## 3.2 Collaboration framework architecture

Many virtual reality applications begin as scene and object visualization environments, having special user interface metaphors for navigation and manipulation, and shown on display devices ranging from CRTs to immersive stereo projection systems. Most science disciplines (and the entertainment industry) use VR techniques to enhance user experiences. As research shows, users *always* desire to share these virtual experiences, either by showing models to prospective audiences, or by having an active remote participation in the environment.

**Table 3.1** *Characterization of Collaboration Features in CVREs*

<b>Network Transmission</b>	<p><b>Distribution:</b> Broadcast, multicast or unicast packages.</p> <p><b>Latency:</b> Considering traffic delays and other perturbations.</p> <p><b>Reliability:</b> Use of positive and negative acknowledgements.</p> <p><b>Bandwidth:</b> As much as possible</p>
<b>Scalable Topology</b>	<p><b>Homogeneous replication using broadcast:</b> Each client maintain a complete replica of the shared environment. Messages across the network maintain state information. No central control; a new client has to wait some time to get information sent by other clients.</p> <p><b>Shared-centralized on a server:</b> Classical client/server model. Shared session information resides at the server. When the server fails it brings down all the clients.</p> <p><b>Shared-distributed with small client/server groups:</b> Several groups of servers and clients. Uses same scheme as mobile phones cells, in which clients are connected to the adequate server.</p> <p><b>Shared-distributed using peer-to-peer actualization:</b> Peer-to-peer connections among all participants, either directly or using a third party relay (broker). Changes are atomically broadcasted to all participants. It comes in two flavors:</p> <p><b>P2Pr:</b> Same replicated scene graph at each node with objects stored locally. Synchronization by using callbacks, and managing of session persistence.</p> <p><b>P2Ps:</b> Sharing objects across the network in a distributed scene graph. Thin replicas shadow remote objects. Network monitors keep shared space correspondence.</p>
<b>Object Complexity</b>	<p><b>Light objects:</b> Short messages containing state, event and control information, require low latency, high-speed transmission rates. (e.g. trackers, sensors, events and status information).</p> <p><b>Remote references:</b> External references shadowing remotely located objects.</p> <p><b>Heavy objects:</b> Medium-atomic data. Big objects requiring reliable transmissions, but small enough to reside in the client memory, (e.g. object 3D geometry, avatars or cameras).</p> <p><b>Real-time streams:</b> Large-segmented data. Data so big it has to be transmitted in pieces and/or continuously, (e.g. big geometric objects, volume information, textures, video, audio, etc.).</p>
<b>Environment Persistence</b>	<p><b>Participating persistence:</b> The CVRE exists only while the participants are in it, and resets when all participants leave the environment.</p> <p><b>Status persistence:</b> The CVRE status is stored elsewhere, to be able to use it at a later time (journaling). Allows the recording of 3D annotations to guide other clients.</p> <p><b>Continuous persistence:</b> The CVRE is always active. A simulation make change scene and objects even if no clients are connected.</p>
<b>User Interaction</b>	<p><b>Interaction strategies common to all CVEs:</b></p> <ul style="list-style-type: none"> <li>– Adequate interfaces for collaborative manipulation and visualization</li> <li>– Teleconference capabilities (streaming video and audio)</li> <li>– Flexible support for data construction</li> <li>– Synchronous and asynchronous collaboration</li> <li>– Adaptive multi-resolution for less sophisticated devices</li> <li>– Standards and interoperability with heterogeneous systems</li> <li>– Replicated or shared spaces</li> </ul> <p><b>Interaction strategies specific of CVREs:</b></p> <ul style="list-style-type: none"> <li>– Use of annotations/indicators for notification of remote events</li> <li>– Multiple users having several views</li> <li>– Use of avatars for remote user representation</li> <li>– Design for low latency response times</li> </ul>

**Table 3.2** *Global Profile of the Studied CVRE's*

<b>Network Transmission</b>	<b>Broadcast:</b> SIMNET <b>Multicast and Unicast:</b> ALICE, Avocado, CAVERN, DIVE, Diverse, DOI, GNU/-Maverik, Massive-3, NAVL, NPSNET-V, REPO-3D, Octopus, VELVET, WOWge, Quake
<b>Scalable Topology</b>	<b>Homogeneous replication using broadcast:</b> Massive, SIMNET, DIVE <b>Shared-centralized on a server:</b> Avocado, CAVERN, NPSNET-V <b>Shared-distributed using small client/server groups:</b> DIVE, Massive-3, NAVL, Octopus, VELVET, Quake <b>Shared-distributed using peer-to-peer actualization:</b> <b>P2Pr:</b> ALICE, DOI <b>P2Ps:</b> Diverse, GNU/Maverik, REPO-3D, WOWge
<b>Object Complexity</b>	<b>Light objects:</b> All <b>Remote references:</b> All but SIMNET <b>Heavy objects:</b> All <b>Real-time streams:</b> CAVERN, Massive-3, WOWge, Quake
<b>Environment Persistence</b>	<b>Participating persistence:</b> Only in navigation mode <b>Status persistence:</b> Massive-3 fully, but all of them implement it to some degree <b>Continuous persistence:</b> SIMNET, but for the rest only needs a daemon simulation running the environment
<b>User Interaction</b>	<b>Interaction strategies common to all CVEs:</b> ALL <b>Interaction strategies specific of CVREs:</b> <ul style="list-style-type: none"> <li><b>Action indicators for notification of remote events:</b> ALICE, REPO-3D, VELVET, WOWge, Quake</li> <li><b>Multiple users having several views:</b> ALICE, Massive-3, DIVE, Diverse, GNU/Maverik, WOWge, Quake</li> <li><b>Avatars for remote user representation:</b> ALL</li> <li><b>Design for low latency response times:</b> GNU/Maverik, NPSNET-V, VELVET, WOWge, Quake</li> </ul>

Evolving collaboration at this stage usually entails the redesign and development of a (new) application, inserting a networking infrastructure under the environment, and other software-porting problems. Issues such as synchrony overheads, concurrent user load and system lags may degrade interaction and adversely affect graphics performance. There are generic API libraries for implementing shared scene graphs [149] that could be used for building multithreaded CVREs, as well as MMORPG engines such as Unreal Engine [58] and Unity3D [133].

The rationale behind our approach is that the object-oriented nature of current stan-



alone VR applications, usually having rendering and user-interface components, would facilitate their transformation into complete CVRE's, by allowing the seamless attachment of a network-based component to enable collaboration.

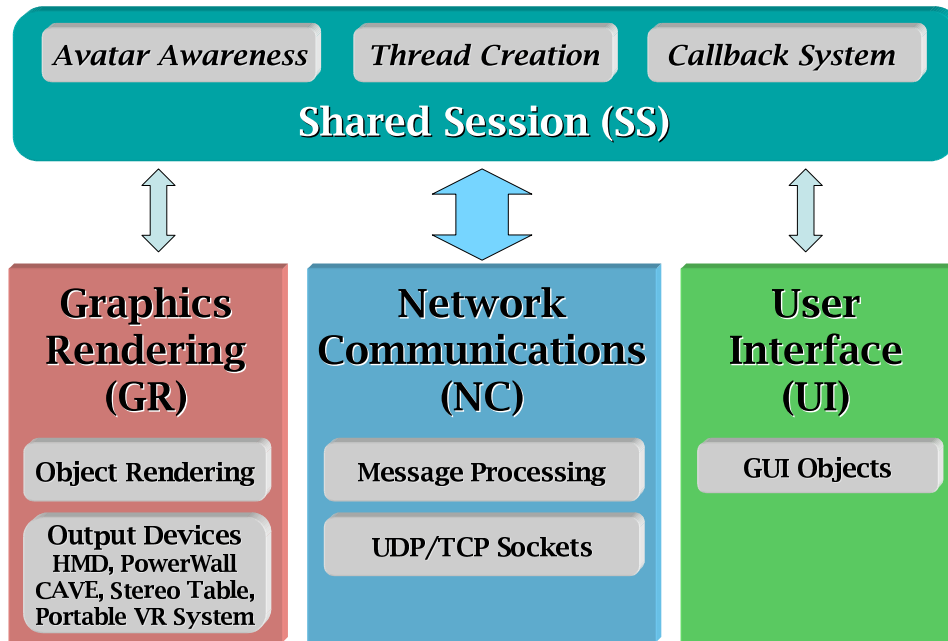
In the following subsections we describe the collaborative features for the proposed superset framework. Given that the different VR tools may spread across platforms and support varied output display systems, the ideal solution should not compromise current designs or imply extensive recoding of components when fitting the collaborative framework. Massive or large-scale implementations were discarded due to user administration performance considerations, although the proposed framework has scaled well for a reasonable number of (less than twenty) participants.

Based on the features described in Section 3.1, our solution involves the implementation of a *multithreaded software components* architecture, a scalable *P2P sharing topology*, a layer implementing *session awareness capabilities*, and a flexible *cross-platform library for network transmission*. We have left for a future implementation the treatment of real-time streaming, since the framework does not modify the current *object granularity* of the target application. On the practical side, it is a portable generic framework, requiring only the instantiation of a custom message interpreting class for the shared session.

### 3.2.1 Multithreaded software components

We assume that a good VR tool is the final product of a sound systems design, developed under a classical MVC paradigm. A standard software engineering practice in Computer Graphics is the refactoring of application objects into at least two weakly cohesive software functional components, graphics rendering and user interface. We decouple the Graphics Rendering (GR) and User Interface (UI) parts and instantiate them in separate threads. The same approach is taken with the new network communications component (NC), launched in its own separate concurrent thread. In this way, advantage is taken of the underlying operating system's context switching, loading the new software components without altering functioning code. This extensible approach allows additional component threads, such as one dedicated to track data acquisition or to interact with haptic devices.

A snapshot of a working framework model is shown in Figure 3.2, detailing each software component. The NC component thread handles communications and message parsing; the top Shared Session (SS) management layer (see the MVCS model in subsection 3.2.3) launches all concurrent threads, tracks users' avatars, propagate state changes to the UI and GR components using callbacks, and is in general responsible for the emerging collaborative behavior; the GR and UI components are mostly untouched except for the binding "glue" to the Shared Session layer.



**Figure 3.2** Collaboration-enabling threaded processes. The framework includes original components (GR and UI), and adds a session layer (SS) with the network component (NC).

This setup is implemented by means of an abstract class wrapper incorporating network awareness and a corresponding message protocol. An appropriate set of mutexes avoid shared state inconsistencies and race conditions when updating information.

### 3.2.2 P2P sharing topology

Fitting any of the client/server topologies would have implied the creation of at least one central server and compromised the applications' standalone behavior. We chose instead a peer-to-peer scalable topology, the most adequate for equal clients with separate access to their models. There are two possible topologies available in the framework: P2Pr [*Peer-to-Peer with scene replication*] and P2Ps [*Peer-to-Peer with scene sharing*].

In a P2Pr topology, each client has its own local scene replica. Since only a few scene objects are modified in the session, collaboration starts as soon as all clients have loaded their common model, and situated themselves within it. If there are no other participants in the environment, it defaults naturally to the standalone behavior.

A P2Ps topology must build a shared scene graph first, with each individual client adding whole chunks. For a particular client, scene graph objects are labeled *local* or *remote* depending on whether they are cached internally or need to be fetched elsewhere. If a client fails, its part of the shared scene must be reconstructed by the others.

### 3.2.2.1 Thin broker for session administration

With no central server, both approaches require a third party to locate clients willing to enter in a session. In our proposal, this third party is called a *message broker*, tracking session interaction, as seen in Figure 3.3. It is loosely based on some CORBA facilities, but without the associated overhead of an IDL implementation. Shared state information is kept through the following services:

- A name service for location and client registration.
- A session management service.
- A session/client state report and mirroring service.

Since the broker is not a bridge, client messages must go directly to their destiny. Each client keeps track of other participants, and periodically may send its current state to the broker for shared session recording purposes.

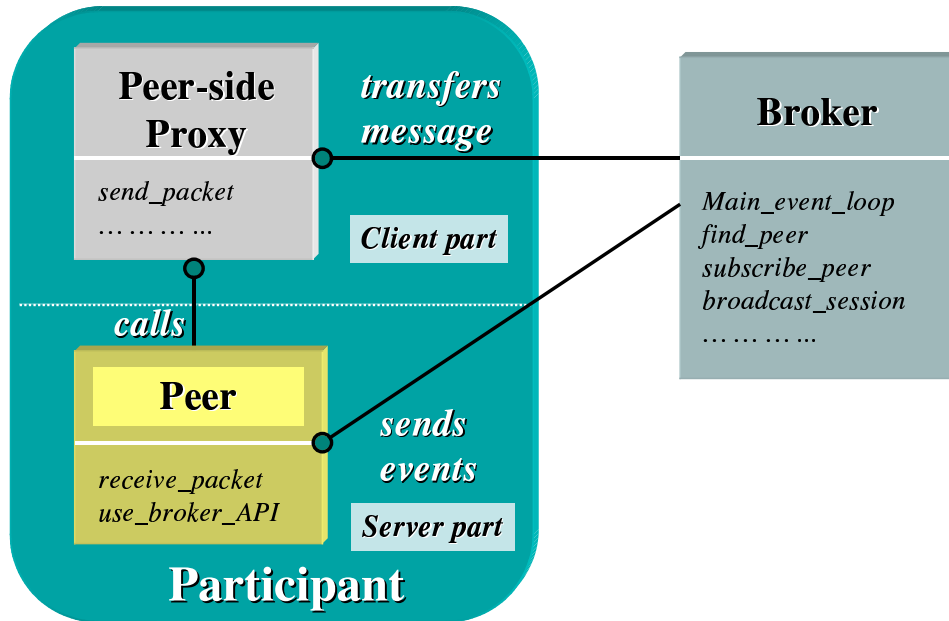
The broker can be easily extended to cover other collaborative functionalities that require some session control over them. An example is explained in subsection 3.4.1 for the control of collaborative manipulations in ALICE.

### 3.2.2.2 Message protocol

The message protocol is short and simple. Its main objective is synchronizing session state across participants. There are two kinds of expected message flows:

- *Continuous Session Updates*, such as participants' position and orientation, moveable objects, video and audio streams; uses fast [or real-time] multicasting.
- *Discrete Session Updates*, high-level changes in object properties (such as texture), manipulation, text chat; uses two channels: reliable unicasting (for peer-to-broker, or P2B) and multicasting (for peer-to-peer, or P2P).

Location and orientation messages may be the *avatars'* camera coordinates being continuously broadcasted among all participants as they move about. Session messages are the ones exchanged between the broker and the peers: connecting and disconnecting, reporting internet addresses and ports, number of active cameras, avatar appearance, global scene file, and other relevant data. Manipulation messages (such as a local client touching, grabbing, adding or modifying an object) are sent to remote users by the callback system to maintain scene coherence among all participants.



**Figure 3.3** Peer-to-Peer Broker class model. Both peers and broker have proxy instances of each other.

A message parser class in the NC thread listens asynchronously in three separate [configurable] ports, one for each of the message channels (continuous location messages, peer-to-broker communications, and peer-to-peer callback diffusion). All messages are of the form shown on Table 3.3

**Table 3.3** Message Parsing Disassembly

<b>MsgID</b>	Session (P2B) or Callback (P2P) ID
<b>Target</b>	Peer IP, Broker IP, All
<b>Flags</b>	Session context properties
<b>Parameters</b>	< parameter, value > tuples

To avoid parsing overhead, no metadata information about the parameters (such as type specification or semantics) is included in the messages. Since participants are all homogeneous, the MsgID determines exactly the expected number, order and type of the received parameters for all peers (and the broker). Thus, no IDL is necessary.

### 3.2.3 Session Awareness Management

After analyzing the desirable characteristics exhibited by existing CRVEs, we concluded that the optimal feature set for session awareness is described by the following:

- Event messaging protocol on network channels.
- Collaborative user interface model.
- Client awareness using *avatars*.
- Session management with differentiated user roles.
- Shared annotation and 3D marker highlighting.
- External real-time verbal communication channel.

For a client in this scenario, there must be perceptual evidence that other entities (human or otherwise) are participating, so 3D client embodiments (avatars) are used to dynamically reflect their position and state in the scene. Clients may want to call others to attention by placing special 3D signals, leaving trails in the scene or modifying the environment. Some users could just browse through the model, while others could have object editing privileges. A collaborative interface metaphor allows the remote manipulation of objects, and session tasks may keep a journaled record of the interaction.

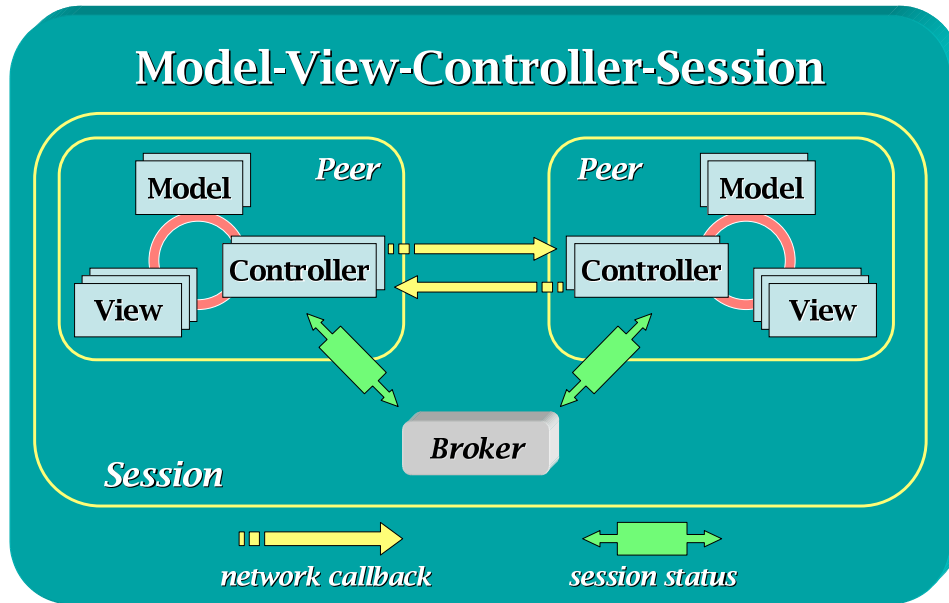
### 3.2.3.1 Collaborative user interface model

The problem to solve when recasting existing VR navigators as CVREs is how to implement the maximal collaborative feature set with the least possible implementation cost, and without affecting the original standalone behavior. We pick from each category of Section 3.1.2 the items that better support awareness under a hybrid Model-View-Controller-Session (MVCS) approach, tying the ALV's *links* as network pipelines to MVC objects, in which:

- MVC objects may not reside together at the same network node, having their Model (structure and behavior) defined at one client, many different Views elsewhere (renderings, at least one for each client), and flow control effected by all. Nodes may have several viewpoints (cameras), allowing for multiple perspectives and resolutions of the same scene.
- Controllers operate using a callback mechanism, routing to the corresponding network nodes for non-local objects, as shown on Figure 3.4. Session layer coherence is maintained by existing network-aware controllers at each node, who also notify the broker. It does not matter whether objects are shared or replicated, so it allows either P2Pr or P2Ps approaches.

### 3.2.3.2 Client awareness using *avatars*

Each client has its own 3D representation traversing the environment, having several active camera perspectives at any time. Avatars broadcast a number of state attributes, such as *position*, *orientation* and *velocity* camera vectors for dead reckoning calculations.



**Figure 3.4** Model-View-Controller-Session (MVCS) objects showing an external broker maintaining session states.

### 3.2.3.3 Session management with differentiated session roles

So far we have identified five different collaborative user behaviors: *standalone*, *peer*, *incognito*, *slave*, and *master*. A *standalone* client is not aware of other clients. It defaults to the original isolated behavior of the application. *Peers* are clients that communicate among themselves using the common message protocol. Users traveling *incognito* may observe scene interaction in “voyeur” mode without other clients knowing it. A *slave* is a peer that is bound to another, correspondingly called a *master*, in the sense that the *master*’s current state is continuously replicated by the remote *slave(s)*. These client roles are voluntary and changeable during a session, leaving open the possibility of adding more roles. A self-explanatory three bit code catalogues their functional role results in the following user codification in Table 3.4 (from left-to-right):

- *bit 2*: whether the client broadcasts messages to others.
- *bit 1*: whether the client listens to remote messages.
- *bit 0*: whether the client binds to another.

**Table 3.4** Bit code for role identification

role	<i>standalone</i>	<i>incognito</i>	<i>peer</i>	<i>master</i>	<i>slave</i>
$b_2b_1b_0$	000	010	110	111	011

### 3.2.3.4 Shared annotation and 3D marker highlighting

Users must not only be aware of each other, they must be able to call the attention of remote participants to some feature or object in the environment. This is accomplished by temporal 3D markers such as arrows, billboards or banners that are pinned at interesting locations.

### 3.2.3.5 External real-time verbal communication channel

Collaborative environments use at least one real-time communication channel to allow the human users behind the workstations to exchange impressions about the virtual experience. The framework does not provide this service, but external suitable open-source cross-platform alternatives such as Gaim, Gnomeeting and others have been used with equivalent ease.

## 3.2.4 Cross-platform network transmission

Since communication is what enables collaboration, the new NC software component handles network communication capabilities. This is done by a cross-platform networking class that allows either datagram-oriented (UDP) and connection-oriented (TCP) communications in IPv4 or IPv6 multicast networks. The NC thread, under a common message protocol, implements the following basic services, each one running on its own separate listening socket:

- Shared event pipeline for sending environment state changes and callback messages.
- Continuous streaming of client data, such as camera position and orientation
- A notifying service for the *Broker*.

When a client reports to the broker, it posts its network address and listening ports. A configurable setup accounts for external firewalling rules, allowing several clients to run concurrently on the same machine by choosing unique port numbers. This enhances performance tests, because it permits the simulation of heavier client loads independently of available workstations. Network traffic is generated only for broker requests, for position or orientation changes, and for shared callbacks (such as object manipulation).

### 3.2.4.1 System synchronization

The framework avoids hosting a central time server by keeping *relative* time differences for every peer-to-peer connection at the client's side. The local event time or *timestamp* is included in each network message. Clients at the other end may process incoming messages as either

- **Immediate:** messages are processed at once.
- **Buffered:** messages are queued by timestamp.

When using the first approach, high network traffic may produce jumpy updates and short temporal inconsistencies. The second is more suitable for replaying events in exact time sequence, at the expense of bigger time delays.

### 3.3 The ALICE Virtual Reality navigator

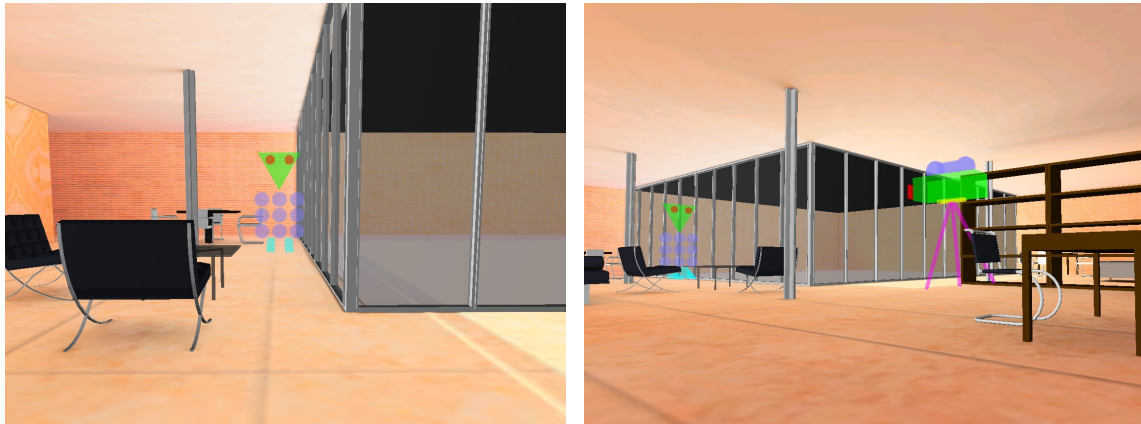
The ALICE VR Real Time Inspector and Navigator [4] is a standalone VR software platform for the real time inspection and navigation of very complex virtual models, developed at the Universitat Politècnica de Catalunya. It has been used in a number of applications such as navigation in urban environments or interior ship design. In order to allow the users of these applications to be able to navigate and inspect complex 3D models in several VR systems, ALICE offers the following features:

- *Stereoscopic visualization*: works either with passive stereo, or active stereo such as Head Mounted Displays [126], and for low cost VR systems [4, 34].
- *User position and orientation tracking*: allowing implicit interaction by following the user's movements and making him feel he is inspecting a real object.
- *Different VR modes of execution*: able to work over different VR display systems like stereoscopic tables, the CAVE, etc.
- *Use of multiple interaction devices*: being able to follow orders from mice, joysticks or VR gloves.

Apart from these external features, ALICE implements internally an extensible system of callbacks and many advanced computer graphics algorithms, in order to be able to work interactively with highly complex scenes. It uses internally a hierarchical object scene graph, keeping also for each element non-geometrical information, allowing, for example, multi-resolution textures. Among these advanced algorithms are the following:

- *Simplification techniques*: ALICE maintains different levels of detail (LoD) for all objects in the scene, allowing a faster visualization of complex models by choosing the right level of detail depending on the distance between the object and the observer [3] (further objects can be visualized with less detail without losing image quality).
- *Visibility culling*: This technique eliminates from the visualization process those parts of the geometric model that will not be visible from the current observer's (camera) position [1]. The technique may be combined in ALICE with the multiple LoDs [2].
- *Collision detection*: The collision detection is a key component for any VR system, being the base for object manipulation (such as using a VR glove to select objects by virtually "touching" them [37]), robotics, vehicle simulators, etc.





(a) The Mies van der Rohe house, as seen from the "camera" avatar

(b) The *incognito*'s viewscene, showing "camera" and "upecito" interacting

**Figure 3.5** *Two Peers collaborate with each other while navigating the architectural environment, as seen from an invisible incognito client's viewpoint.*

### 3.4 Performance analysis of the framework

The ALICE application is already factored into two software components, Graphics Rendering and User Interface. The User Interface component is provided by Qt, an object-oriented user interface cross-platform toolkit (using the MVC paradigm) for MS Windows, several UNIX variants, GNU/Linux and MacOS X.

The decoupled callback hooks system in ALICE connects user events to the graphics pipeline by means of a indexed command list. Each element of the lists stores a settable reference (the "hook") to some object's method (the "callback" function). When an UI event triggers a particular command, its corresponding callback hook is executed with the provided event information and current environment state.

#### 3.4.1 Framework validation in ALICE

Given all the above, it was considered a suitable candidate for enhancing its collaboration features. Just changing some flags at the compiling phase allows the UI and GR components to load in separate threads. Next, the following steps were taken to fit ALICE into the framework:

1. Instantiate the shared session (SS) layer class, holding all common state awareness attributes, such as the scene graph, avatars, remote references for the broker and the list of participants.
2. Choose a scalable topology (P2Pr, for this version).
3. Devise the *peer-to-peer* and *peer-to-broker* message protocols.

4. Instantiate the message parser class to process event messages, and place it in the networking communications (NC) component.
5. Wrap the GR, UI and NC software components as SS layer class attributes, and launch each of them in a separate thread.
6. Add one method call to provide a callback hook linking the message parser class in the NC component to the session-update method of the SS component.
7. Add one method call in the UI's main method to provide a callback hook to the SS layer.
8. Add one method call in the GR to provide the callback hook syncing the cameras and states of network peers just before rendering.
9. Instantiate the broker class, adding the necessary services.

The whole setup comprises just five classes: session-peer-broker communications, thread management, message parsing, callback serialization and remote camera handling.

#### 3.4.1.1 Peer and Broker instantiation

A scalable P2Pr topology was initially chosen [137], given that all clients already function with local scene replicas and it would not change much ALICE's behavior. In shared mode, the broker indicates the remote reference of the current scene, so hopefully everyone would be placed in the same model. The broker must be active for a session to be initiated by at least two subscribing participants. Each client may choose a session role (usually *peer*) and an avatar representation (from a menu), as shown in Figure 3.5, while keeping a list of the current interactions with other users. As they navigate, clients may chat to each other, or place 3D markers to call attention to some feature.

Clients can also take the role of “voluntary slaves” for some other user, which now becomes a camera server. The *slave* shuts down its own cameras and reflects the *master's* camera viewpoint and actions, the latter effectively taking possession of the slave's remote display devices. This feature may also be used to “teleport” a participant to the position of another, which is very useful to avoid losing virtual eye-contact among peers. Each node does independent renderings, which allows a client to show a wireframe representation while another fully renders the same scene.

#### 3.4.1.2 Message protocol implementation

The message protocol for communicating with the broker is the same for all implementations, since it is application-independent. The continuous communication channel for location/orientation messages may be used as implemented for avatar presence notification, although it may be extended to add continuous streaming of data acquisition from haptic input devices. The message parser class must be extended to handle the manipulation messages proper to each new VR

tool. Callbacks that affect model integrity or are subjected to user interaction (such as a local client touching, grabbing, adding or modifying an object or the scene) were fitted with serialization and de-serialization methods to encode and decode the callback replication message, which is immediately multicasted to all peers in the session (plus the broker, if the session is being recorded). Out of the growing callback set of ALICE (around 100), only a subset of 14 affect model integrity, shared scene state and object appearance, although more may be defined in the protocol.

Semaphores in the NC thread activate and deactivate the UI and GR threads when it is modifying incoming packets such as clients' cameras, so to avoid the race conditions so common to concurrent programming.

### **3.4.2 Emerging ALICE collaborative features**

A practical side arising of an implementation based on abstract wrapper classes, is that it is platform-agnostic and extensible, which makes it quite portable. Each application only needs to inherit from the message parsing class, add its own protocol processing code and provide the hooks for the UI and GR components. Given the new features provided by the framework, ALICE was enhanced visually and structurally with three new important additions: (1) remote user windows, (2) extended callback mechanism, and (3) collaborative object manipulation.

#### **3.4.2.1 Remote user windows**

This additional feature to the collaborative capabilities of ALICE allows the user to overlay several small frames echoing the rendering of remote collaborating users on top of the application's own rendering. In Figure 3.6 the main window shows the avatar of the remote user whose rendering is being visualized in the small north-east corner.

Since the purpose of this feature is to know how the remote user is seeing the scene at any time, some information about the local rendering of that user is also required to pass through the network (apart from the information of position and orientation of his camera). This information includes, for example, the position of far and near clipping planes.

#### **3.4.2.2 Extended callback mechanism**

When a user interacts with ALICE, he can cause some changes in the scene or in his representation in the collaborative session (changing the avatar or becoming stand-alone, for example). These changes should be broadcasted to the rest of the participants on the session, so they can be aware of what the changes were.

As already stated before, an ALICE callback is an code entity hooked to a certain task that has to be done in the application. Callbacks that imply a network communication, causing the replication of its task in all other participants in the collaborative session, require the implementation of methods for serializing, sending, receiving and de-serializing the information associated with it.

Since each callback may involve different number and type of parameters, its serialization and de-serialization is known and performed by the same callback, at both ends of the network. This also has the advantage of packing the parameters as one stream of data, avoiding sending redundant metadata overhead. The only information that has to be sent with the callback network message is the unique ID of the callback method able to de-serialize the message.

Examples of callbacks included in the collaborative ALICE application are the following:

- *change avatar representation*, the user selects a different representation for its own avatar;
- *add a new light to the scene*;
- *select/deselect an object*;
- *change a property of the selected object*;
- *eliminate the selected object*;
- *move/scale/rotate the selected object*;

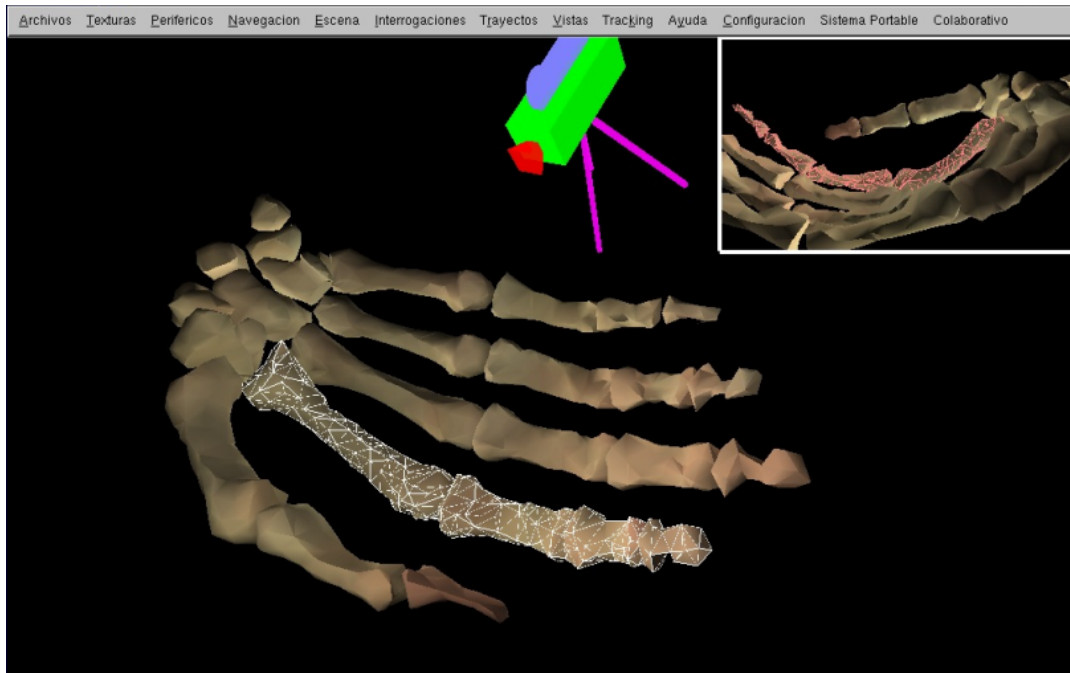
The last four callbacks listed are directly related with the collaborative manipulation of objects explained next.

### 3.4.2.3 Collaborative object manipulation

ALICE, as a navigator and inspector, has limited modeling options in the scene, but it allows the user to move and delete objects and also to change several object attributes like color, for example.

Since this collaborative object manipulation requires some control at session level, the broker's framework (explained in subsection 3.2.2) has been extended to support a locking hierarchy for object selection. When a user wants to manipulate an object he has to select it first by asking the broker whether the object is already selected by another user. The requests for selecting an object arriving to the broker are served by timestamp, so the oldest request is the one who locks the selection. When the broker accepts a selection coming from a user, this selection is multicasted (via the callback) to the rest of the users in the session.

In order to increase user awareness of other participants, objects held by remote users are labeled as *remotely selected* in the local copy of the scene, and the visualization routine overlays a *red* wireframe around them. Objects labeled *locally selected* are overlaid with a *white* wireframe. This information of remote selections is also kept in the list of remote users, to be able to recognize the user currently holding a certain object, useful when performing the rendering of that user's viewpoint in the remote frames (for example, in Figure 3.6 the finger being manipulated by the local user is being shown as *remotely selected* in the small frame of a separate peer).



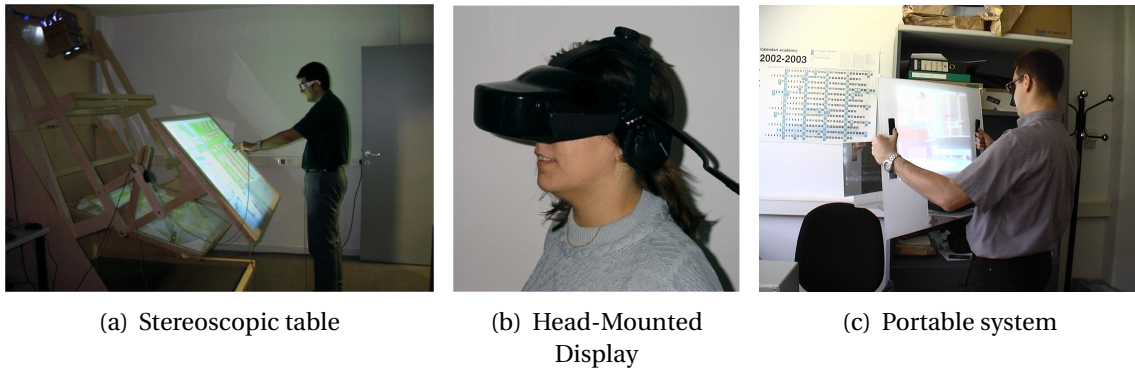
**Figure 3.6** A main window with a remote client visualization on the upper-right corner.

### 3.4.3 Evaluation of results

ALICE has been successfully used as a collaborative platform over different VR systems, such as HMDs, Stereoscopic tables [4] or a portable system [34] (see Figure 3.7). Some of the applications are the following:

- *Inspection and navigation in the interior of a ship.* Ship designers cannot use real prototypes to evaluate their designs, so they need virtual prototypes to be evaluated. Different users may discuss about the resulting design without needing to be at the same place at the same time.
- *Training in medicine.* A surgeon can show the students, for example, how the scalpel should be used for a certain surgery incision.
- *Inspection and modification of architectural design.* The virtual design made by the architects can be shown to potential clients in order to know their opinion. A client, by collaborating with the architect (see Figure 3.5), can also make small changes such as moving furniture or walls in order to propose other possibilities not far away of the initial design.

We have tested ALICE's remote collaboration and navigation services in the several VR systems in our lab, and also in sessions with the Girona University (located 100 Km. from the Barcelona campus) through a 10Mb wide area network connection. In our lab we have available HMDs, a stereoscopic table, a CAVE, a MiniVR system and flat displays; and a similar setup at the Girona campus. The results obtained from our tests can be seen in the following table. The scene used on these



**Figure 3.7** *Some of the different VR systems (commercial and homegrown) where ALICE has been tested.*

tests (the interior of a ship, see Figure 3.8) contains 50.000+ polygons (close to 300.000 rendering triangles), but on purpose does not have complex textures that could skew graphics performance. Table 3.5 shows the results obtained in the communication of 2, 4 and 8 workstations using unicast addresses from both sites.

**Table 3.5** *Collaborative scene performance measurements*

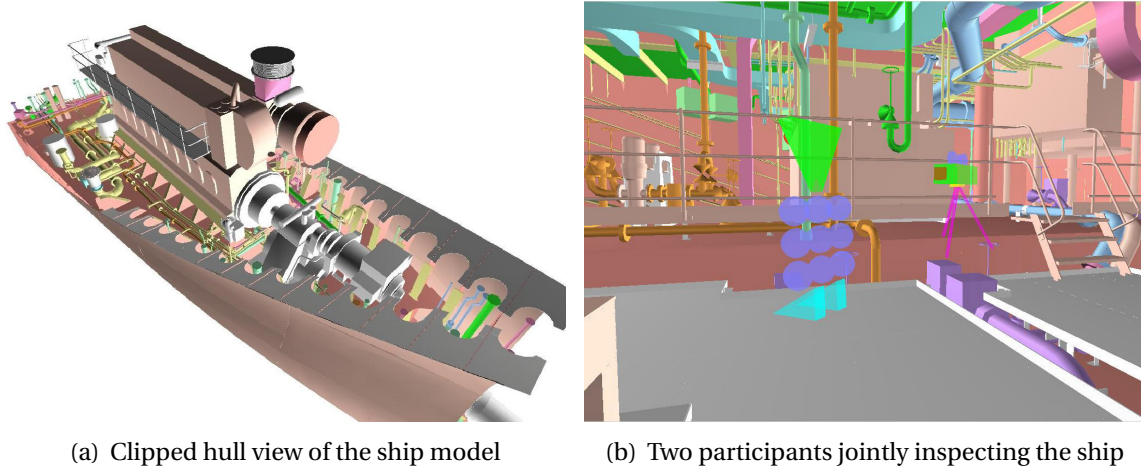
Participants	1	2	4	8
Avg. Number of messages	–	2539	8067	14331
Avg. Total Net. Time (ms)	–	35	31	46
Avg. Roundtrip Time (ms)	–	13	46	57
Avg. Framerate (f/s)	47.3	45.2	44.2	42.7

In the table we observe the average total number of messages sent through the network in a series of repeated navigation trials, each test lasting 4 minutes. The total network time (in milliseconds) gives information about how much time ALICE spent in the transmission of messages during these 4 minutes tests (this means that only around 0.1-0.2% of total time was spent in network communications). The roundtrip time is also indicated in milliseconds. Since for this test we use unicast addresses, roundtrip time increases as more peers participate in the session. Finally, the table shows the average rendering framerate achieved for each case, which indicates that increasing the number of nodes affects graphics performance very slightly compared to the standalone performance, and is comparable to similar setups in the studied environments.

As already stated in subsection 3.4.1, the migration of ALICE to a CVRE was fast and uneventful. Based on the fact that the application was already designed considering graphics rendering and user interface as separate components, its porting to our framework only required to define an adequate message protocol, connect the appropriate callback hooks, and add two method calls and corresponding code hooks in order to attach the application to the new network and session parts.



Following the same migration scheme, it should be easier to transform any other VR application into a collaborative VR application, given that the peer-to-broker protocol won't change much, if at all. In fact we are presently porting another application built in our lab which addresses inspection and management of complex medical and organ models. Some fine-tuning must be performed to



**Figure 3.8** *Outside and inside views of the Texaco tanker model used in the performance measurements.*

adjust threaded execution. A highly textured model may take a while to render, making timely interaction slow and difficult. Although this can not be avoided, it may be reduced by changing thread priorities to model complexity and network traffic. To measure performance in a clogged network, we created high traffic conditions by loading multiple ALICE instances at each workstation, resulting in clients falling out-of-sync due to packet losses. To minimize these latency problems, there is an option to process only the most recently received (by timestamp) packet from each peer in the environment, at the expense of a somewhat jumpier navigation.

### 3.5 Conclusions from this chapter

Having characterized the generic collaborative features for VR systems, we have developed an agile framework to add collaborative capabilities to standalone scene renderers. We incorporated a hybrid distributed user interaction model, based on multithreaded software components, peer-to-peer scalable topology for network communications, a custom protocol with message-passing channels, and a multi-camera subscription model with interchangeable user roles.

The generic crossplatform framework was validated by a fast porting of the ALICE VR Navigator, allowing an easy evolution of (formerly stand-alone) VR applications into complete collaborative virtual reality environments. The proposed mechanism for camera management and sharing is an

easy one to learn for users and seems to be adequate for remote presence and collaboration tasks.

The next step will be extending collaborative breadth of the framework by including in the Session layer a fourth thread, one dedicated to handle haptic devices, adding high frequency force-feedback event sampling to the interactive session repertoire. To this end, given the huge scene size of current VR scenes and objects, we plan to use the framework for developing collaborative applications with a peer-to-peer with sharing scheme (P2Ps), and also allow the incremental streaming of multi-resolution objects to improve rendering performance and scalability, with special attention to image-based speeding techniques that reduce geometric density of models without degrading haptic perception of detail. We will set a battery of experiments with untrained users soon in order to have a more accurate measure of perception and ease-of-use.

The corresponding research and its results have been recorded and described in the following references: [136], [137], [134], and [135].





## *Per-face heightfield haptic rendering*



*Stop trying to hit me and hit me –*

*Morpheus, The Matrix*

**I**N the following sections we develop a new treatment for haptic perception of fine detail, postulating a method that dresses triangle meshes with image-based composite mesostructures “coats”, built out of heightfield displacement textures and normal maps. These mesostructure coats are used to create, enhance or substitute surface features in low, mid and higher frequencies, adding non-existent detail at a very low processing cost. We then delve into explaining the set of usability tests that allow us a fair comparison of rendering techniques using the same mesh models and surface details. This allows us to measure quantitative differences on perception, performance, and suitability for surface fine detail, and also to determine the limits in which the proposed solution serves its purpose.

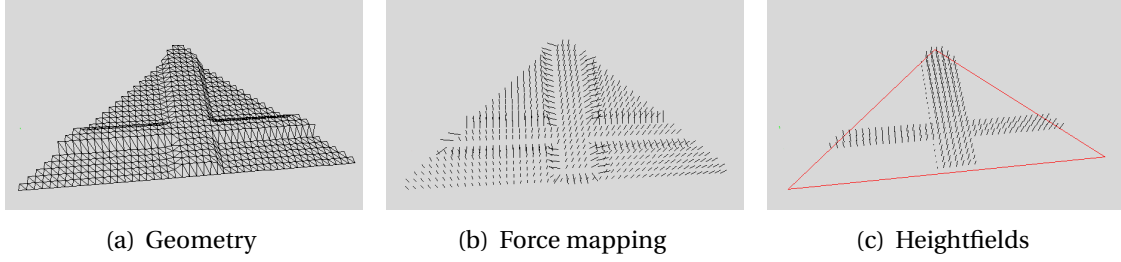
Our objective is to find an algorithm to allow haptic perception of surface details in very complex scenes represented by triangle meshes, and to compare it to other known solutions.

The experimental setup is based on an algorithm for the generation of a detailed geometry, a force mapping algorithm based on normal maps, and the solution proposed which incorporates heightfield mapping (see figure 4.1).

### **4.1 Models for haptic perception of surface details**

The simulation of surface details in very complex models has not been a problem from the visualization point of view since the late 70's. Algorithms such as the use of colored textures or bump-mapping are well known in the literature [9]. An image-based visualization approach uses impostors to replace the nearest object geometry, as described by Policarpo [102] and Baboud [5] for fast shading of geometric objects using displaced-mapped impostors, either as a two-sided back/front map or a six-sided cube map.

In the case of haptic perception, as we have seen in Section 2.3, any simulation algorithm should be efficient enough to achieve the high frequency updates (1000 Hz) required by the force feedback devices.



**Figure 4.1** *Different approaches to simulate surface details in haptic perception.*

#### 4.1.1 Generation of geometry

A direct algorithm simulates surface details by generating the local geometry corresponding to these details. This geometry can be calculated as a preprocess for the whole model, or locally for each triangle of the mesh at run time, by deciding which triangles of the mesh are inside some bounding structure enclosing the haptic probe.

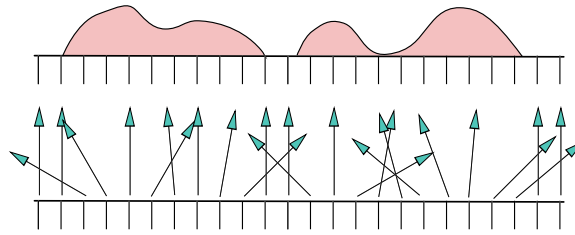
A collision detection algorithm then tracks the haptic probe (or *god-object*), which is represented in the screen by a *proxy*. When moving freely in empty space, its coordinates coincide. When the *god-object* hits a triangle, the *proxy* clamps to the collision point, and a force is exerted in the haptic device to push the *god-object* to the nearest point in the hit triangle, with the *proxy* following suit along the surface.

In the aforementioned cases, the cost in time and/or space of modeling detailed surface using just geometry is not affordable given the frequency updates requirement. Having many small triangles, the haptic probe may well be at another point in the surface (a far-off triangle) after detecting a collision, and the resulting force interplay can cause unstable behavior, adding spurious force feedback, inducing vibration or simply wrong sampling (haptic aliasing).

#### 4.1.2 Force mapping

One of the algorithms used in visualization to simulate surface details is the *bump-mapping* algorithm. It consists of a texture map where each value corresponds to a normal vector direction (RGB corresponds to XYZ coordinates respectively), which can be different for each point in the map. A discretization of the triangle's surface is achieved by mapping texture points to corresponding points within the mesh triangle, having a different normal vector at each point. The visualization *bump-mapping* algorithm uses these normal vectors at the discrete points to calculate illumination values inside the triangle, simulating the surface details.

The haptic perception *bump-mapping* algorithm, also called *force mapping* [7] (see listing in Algorithm 4.1) is based on the same principle. It uses the normal vector at surface points (shown in figure 4.2), either procedurally or sampled from a texture map, to calculate force direction and magnitude to be applied to the haptic device after it collides with a triangle.



**Figure 4.2** Correspondence between simulated surface (top) and normal vectors (bottom).

---

**Algorithm 4.1** Haptic force mapping.

---

```

1: loop                                     ▷ Sample new haptic probe position
2:   Detect collision with the mesh triangles
3:   if ( $\exists$  collision) then                 ▷ Found a collision point within triangle
4:     Calculate repulsion force direction by sampling
5:       point surface normal stored as a texture map.
6:     Apply this force to the haptic device
7:   end if
8: end loop

```

---

By using this haptic perception algorithm and implementing the *bump-mapping* visualization as a GPU shader, we achieve a correct perception of surface roughness when height distortion is not very steep. This is because the collision is always detected against the triangle of the mesh, so a perception of a displacement upward or downward from the triangle surface is not possible (more detailed results in Section 4.4).

## 4.2 Per-face heightfield haptic rendering algorithm

Heightfield rendering has been used primarily as a technique for imaging large datasets out of GIS data. These are usually huge files stored as  $\text{RGB}\alpha$  images, with the RGB providing texture information and the  $\alpha$  channel providing the vertical height. A grid is used to scan the heightfield, interpolating and filtering values as needed, and creating a corresponding geometry.

For haptic rendering in this sort of applications (terrain models), the *proxy's* calculated position must correspond accurately to that of the tracked *god-object's*, because the magnitude of the heightfield map is usually very big and small differences may cause wrong perception. These models generally render a whole terrain as a gigantic heightfield map [103].

The requirements can change slightly when the objective is not a GIS application. In our case, our interest is using heightfield mapping applied to haptically render a triangle mesh model, in which different heightfield textures can be applied to different triangles. Another requirement in

this sort of applications is that the triangles are not as big as the whole model, and actually in many cases they are small enough to be represented in a small area of the screen. This implies that the accuracy required for computing the *proxy*'s position as explained above is not so strict in this case, since we want to simulate small surface details in a *single* triangle of the mesh. On the contrary, it becomes much more important to be able to determine, at a low cost, which triangle is being collided by the haptic probe, so the haptic render algorithm can apply the appropriate heightfield mapping.

In order to fulfil this requirements our algorithm combines different techniques to make this haptic rendering as efficient as possible:

- The triangle mesh is represented by an octree space decomposition, which allows a very quick detection of the area where the haptic probe is located, discarding early in the process a very high amount of geometry of the model which does not need to be considered in the rest of the algorithm.
- The collision with a triangle of the mesh is detected against a prism created between the triangle and a copy of the triangle displaced over a certain maximum distance for the heightfield map values (see figure 4.3(a)). All triangles created to define this prism share the same identifier with the original surface triangle of the mesh, so the computation of the right triangle is immediate.
- A potential collision against any side of the prism triggers the heightfield haptic rendering of a small grid cell just lying in a perpendicular triangle under the probe, shadowing the probe's movement. If at any time the probe descends below the signaled height for that cell, a repelling force is applied to the haptic probe along the surface normal at the projected point, proportional to the height difference (or penetration). This forces the god-object onto the surface at that point, at which the force ceases to be (see figure 4.3(b)).
- All the heightfield maps are forced to distance zero on the edges of the mapped triangle, avoiding discontinuities on these edges that can cause instability in the haptic device.

Algorithm 4.2, used for this approach, works as follows, shown also in Figure 4.3:

For haptic rendering, the *god-object*'s location and orientation are tracked as a the vector or ray starting from its last sampled position to its actual whereabouts. As before, there is an underlying base grid, in which the heightfields may be represented as triangular, square, bilinear or NURB patches.

A local search is started to see whether this ray intersects one or more of the patches (depending on the angle, it may go through quite a few of them), and select the closest one with some distance function. After that, it behaves as in the geometric haptic modeling, clamping the *proxy* to the surface and pushing the haptic probe to the surface, which again may not coincide with the initial collision point.

**Algorithm 4.2** Heightfield-displacement rendering.

---

```

1:                                     ▷ Optimal sampling speed between 200-1000 Hz
2: loop                               ▷ Sample haptic probe position  $P_H$ ;
3:    $P_H = (x_h, y_h, z_h)$ ;
4:   Detect potential collisions with triangles in the mesh;
5:   if ( $\exists$  collision against prism's triangle  $T_j$ ) then           ▷ Haptic Probe  $P_H$  is inside  $T_j$ 's prism
6:                                     ▷ so we must check against haptic texture;
7:     Project  $P_H$  against  $T$  obtaining surface point  $P$ ;
8:     Compute 2D texture coords  $(s, t)$  of  $P$  over  $T_j$ ;           ▷ Sample the heightfield displacement
9:      $Z = H(s, t)$ ;
10:     $penetration = Z - distance(P_H, P)$ ;
11:    if ( $penetration > 0$ ) then                                       ▷ Positive penetration, a real collision;
12:      Calculate repulsing force for the device;
13:      Compute  $\vec{F}(penetration)$  along the normal  $\vec{N}_j$  of  $T_j$ ;
14:       $\vec{F}(penetration) = k \cdot penetration \cdot \vec{N}_j$ ,  $k$  constant;
15:      Apply  $\vec{F}(penetration)$  at the device;
16:    end if
17:  end if
18: end loop

```

---

### 4.2.1 Comparison between force mapping and per-face heightfield haptic rendering algorithms

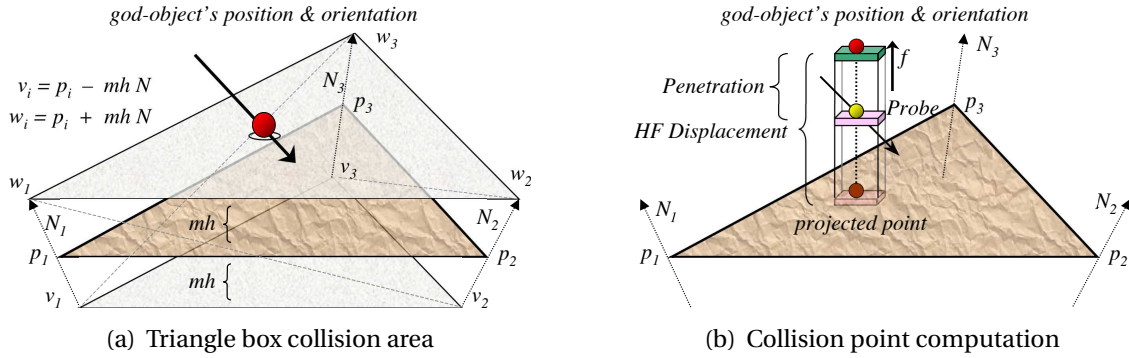
In order to compare the quality of haptic perception using both force mapping and height fields, we decided to set up the environment experimental conditions as close as possible for each run. This means using the same base models for both runs, and an exact correspondence between the relief map used as a heightfield and the normals map used for force mapping.

First, a sphere, built recursively at several resolutions out of a perfect icosahedron was chosen as the base model. For the sake of enhancing the features detailed in the article, it has been set to the simplest possible resolution for the figures appearing in this article, to better appreciate visually the rendered surface texture.

Secondly, we generated synthetic heightfield maps, and used it to compute a corresponding normal map, using a standard GLSL shader. In this way we obtained corresponding pairs of heightfield and normal maps representing the same 3D geometry, as shown on figure 4.4. In the end we settled for the three textures shown here, since each allows to perceive different surface characteristics.

The textures chosen are an embossed cross with only flat and vertical surfaces, gently sloping circles with a soft gradient, and a texture with alternating polished and variable bumpy areas. Additionally, taking advantage of the graphics card hardware, we implemented the shading part corresponding to our bump mapping as a GPU shader, just to allow a better haptic sampling rate.

We show on figure 4.5, different renderings using force mapping and heightfields. It is evident that bump mapping makes a smoother visualization, and it is much faster, even without the shader. However, in terms of haptic perception, the comparisons are quite different and depend on the

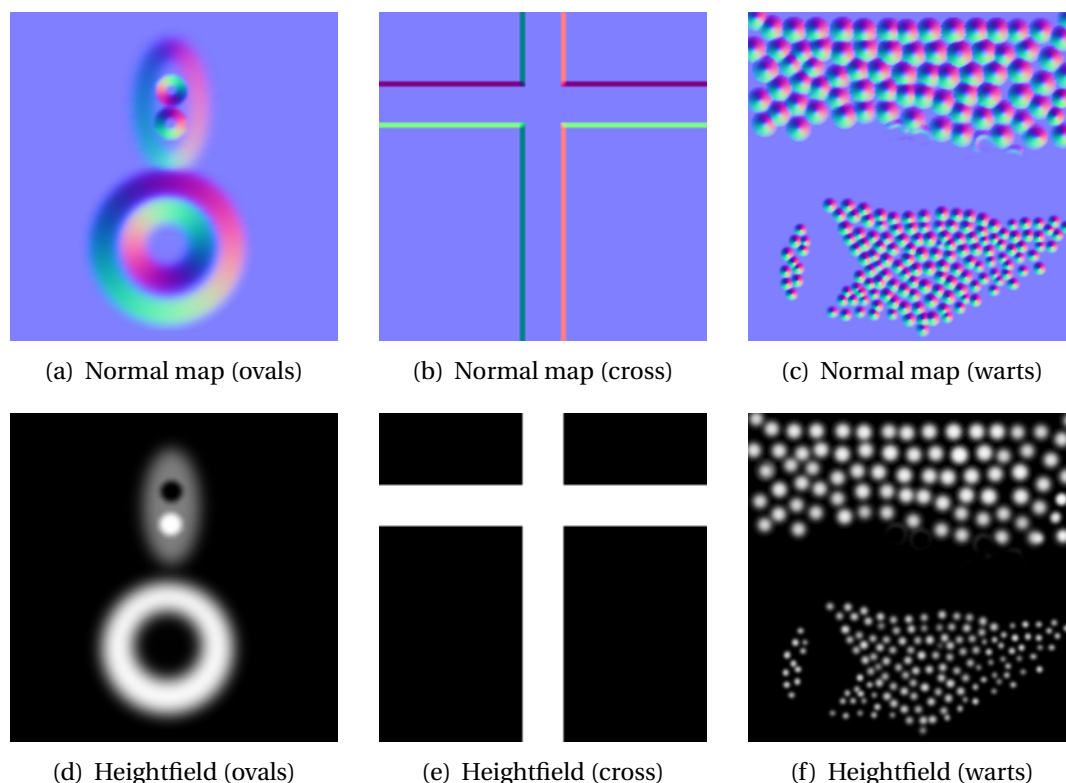


**Figure 4.3** Heightfield collision mapping: once the probe is within a prism, the force's repulsion normal is sampled relative to position, and the force magnitude response is proportional to the penetration.

characteristics of the texture map.

We studied the haptic perception of the user in both methods, force mapping and our heightfield algorithm, with several texture maps with different characteristics:

- In the case of a texture map like the one shown in figures 4.5(e) and 4.5(f), where the texture image shows two bumpy areas, the resulting perception is somehow similar in both methods. The user perceives a certain roughness in the fine bumpy area and a bumping feeling and certain guidance among bumps in the coarse bumpy area.
- In the case of a texture like the one shown in figures 4.5(c) and 4.5(d), where the texture image shows a big oval and two small bumps over the surface, the resulting perception is a bit different between the two methods. In the part of small bumps there is almost no difference, but in the big oval part, although in both cases when the user is inside the oval there is some similar resistance to go out, with the force mapping method the user only perceives the resistance for going up to the oval while with the heightfield method the perception is clearly going up to the oval and down from it.
- In the case of a texture like the one shown in figures 4.5(a) and 4.5(b), where the texture image shows a cross step over the surface, the perception is clearly different between the two methods. With the force mapping method the user only perceives resistance on the going up and a jump going down, but no height differences can be perceived. Moreover, there are some instabilities on this perception in the areas of the middle of the cross, because there are neighbour points with very different normal directions. With the heightfield method the user perception is much better in this case, because the parts of the cross going up and down give the feeling of going up and down with different height on the top of the cross than on the base. There is no instability either in those areas in the middle of the cross.



**Figure 4.4** *Normal and height maps of choice test surfaces.*

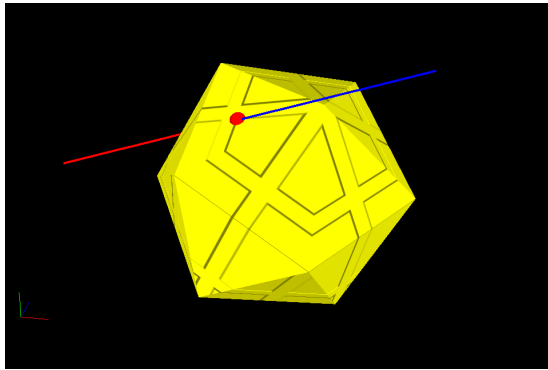
As a summary, we can say that the force mapping method can be a good approximation for modeling an apparent roughness of material, but is not sufficient for irregular normal maps where the perception has to be tight to the texture shape. This problem is solved with our height field algorithm, which gives an accurate sense of the surface characteristics.

However, the perceived sensation at the end of the probe does not quite convey a sense on the surface, and although gives a good approximation for modeling the apparent roughness of the material, it is a turbulent ride when using very irregular normal maps, such as those arising from terrain modeling, and other textures that need a sense of going up or down, even at small steps.

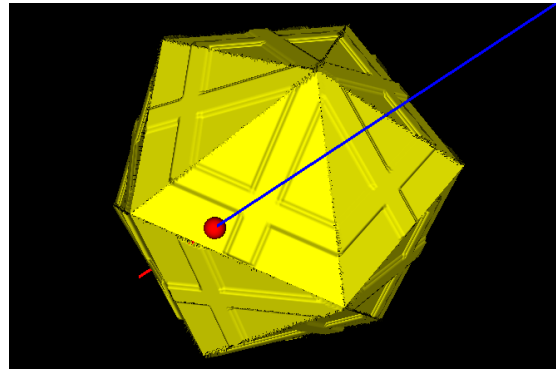
On the other hand, rendering heightfields, as in the approach presented, gives an accurate sense of the surface characteristics, allowing 6 DOF movement while keeping the computing cost relatively low, as long as the underlying textures below the triangles are of a reasonable size. Individual local heightfields with texture sizes between 128 and 256 pixels per side work reasonably well. Anything lower feels coarse and higher texture densities may be too much, either affecting haptic sampling rates, rendering framerates or both.

The equipment used is a HapticMASTER from FCS, having a built-in wide haptic 3D space, able to exert forces from a delicate 0.01 N up to a very heavy 250 N, with a wide 3D perception field, equivalent to a wedge of 40 cm x 36 cm x 1 radian. The PC is a 2GHz Pentium IV with an ATI Radeon 9700 graphics card.

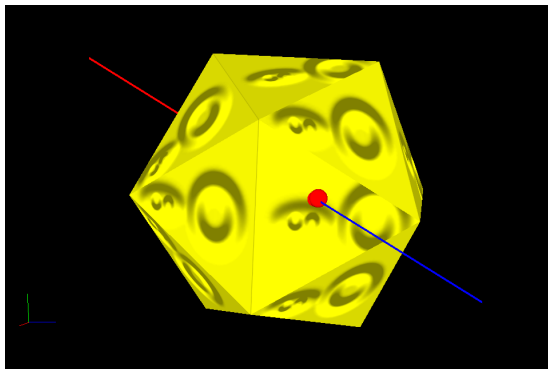




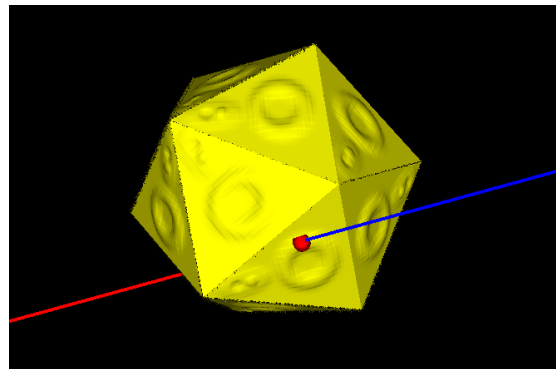
(a) Cross (using force maps)



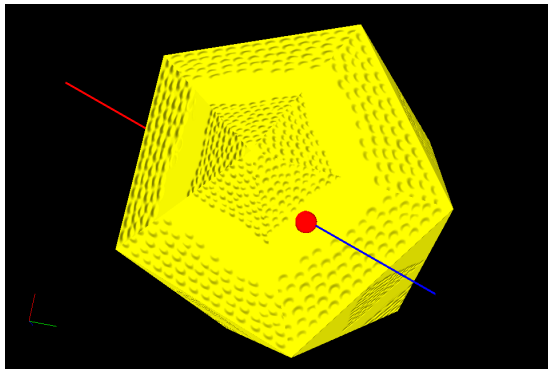
(b) Cross (using heightfields)



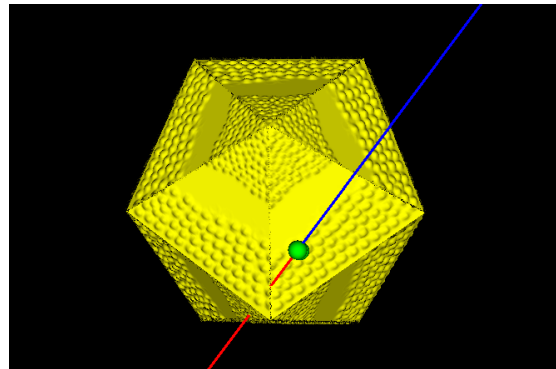
(c) Ovals (using force maps)



(d) Ovals (using heightfields)



(e) Warts (using force maps)



(f) Warts (using heightfields)

**Figure 4.5** *Force-mapping vs. heightfield haptic collisions.*

By modulating the underlying texture by subsampling or gaussian smoothing of texture patches, the same surface can be rendered as grainy detail, a softer sloping surface or blocky one, altering the perceived qualities of the same triangle mesh. There can be several texture maps active and rendered at any time, one for each base triangle.

In the following sections we develop a new method for haptic surface perception, overlaying

simple triangle meshes with image-based composite mesostructures built out of heightfield displacement textures and normal maps. A devised usability testing trial set allows making fair comparisons of rendering techniques using the same mesh models and surface details, measuring quality and performance of the proposed solution for the perception of fine surface detail.

### 4.3 Mesostructure model for haptic rendering

We proceed now to elaborate on a method that proposes a global solution to the afore mentioned problems. Instead of applying the force in the normal direction of the base triangle  $T_k$ , a more accurate rendering approach applies the repulsing force in the exact direction of the normal at the specific impacted surface point. Normals are generated directly from the heightfield displacement texture and both are stored in a texture, creating what we call a Hybrid Rugosity Mesostructure or *HRM*. Taking into account the traversal direction when touching a surface, the haptic point is pushed in the direction of the normal, and a constraint system combines this repulsion with the force exerted by the user at the probe, producing a change of position and orientation without incurring in lagged responses.

This allows to vary surface sensation exploration by “coating” a mesh with several surface reliefs at different frequencies. This image-based general procedure, shown in Algorithm 4.3, uses the previously defined prisms with an added twist. The HRM tuples from normal maps and heightfield displacements  $[\vec{N}(s, t), H(s, t)]$  correspond to an  $\text{RGB}\alpha$  texture, having coordinates  $\langle r, g, b, \alpha \rangle = \langle N_x, N_y, N_z, H_w \rangle$ .

The heightfield-normal tuples may be provided as static or procedural 2D, 3D or 4D (3 + time) textures, allowing for even greater complexity of haptic perception and correspondence with visual renderers. Friction, viscosity, magnetism, color and other surface properties may be easily added as additional entries on the HRM, requiring only the modification of the force-response accordingly. Haptic resolution gets scaled in sync with the current visual zoom state, so features not measurable at lower zoom levels (“blurred”) become noticeable at close range, and surface perception becomes more accurate.

#### 4.3.1 Blending haptic mesostructure at the edges

Algorithm 4.3 also computes soft transitions at triangle edges having different mesostructures using a simple interpolation scheme. For each face in the mesh, we keep track of neighborhood information of all adjoining face indices. When following along the surface of the mesh, the mesostructures in neighboring faces may produce an abrupt topographic change at the edge, that if left to stand will produce a sudden force kick (in magnitude and orientation) in the haptic device. To eliminate these abrupt jumps, we devised the following stitching procedure to blend the transition among convex faces. Heightfield and normals closer to the edges are sampled from the rugosity mesostructure

**Algorithm 4.3** Haptic mesostructure-blended rendering.

---

```

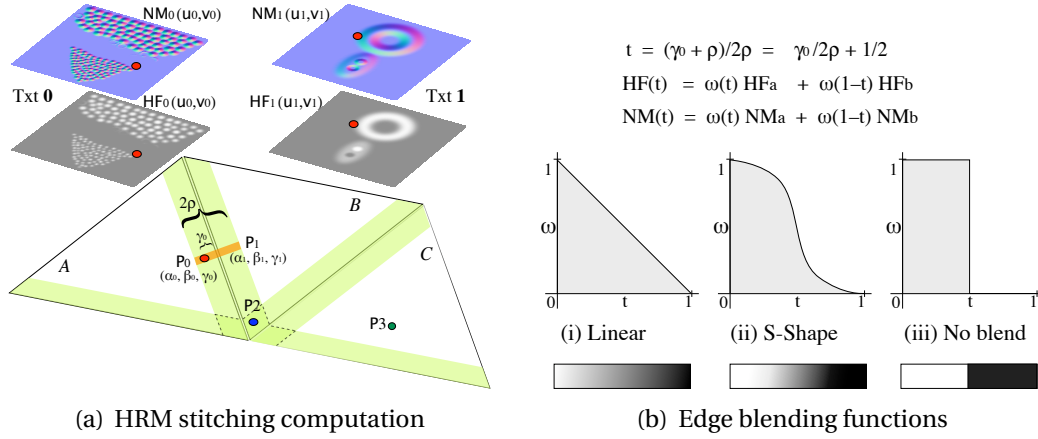
1: loop
2:   Sample haptic probe position  $P_H = (x_H, y_H, z_H)$ ;           ▷ Detect collisions against the triangle octree;
3:   if ( $\exists$  collision at some triangle prism  $T$ ) then           ▷ The haptic probe is inside the prism
4:     Project  $P_H$  against  $T$  to obtain surface point  $P$ ;
5:     Compute 2D texture coords  $(s, t)$  of  $P$  over  $T$ ;
6:     Obtain barycentric coords  $\alpha, \beta$ , and  $\gamma$  of  $P$  in  $T$ ;
7:     if ( $\exists \alpha, \beta$ , or  $\gamma \geq 1 - \rho$ ) then           ▷ We are within  $\rho$  distance of an edge
8:        $\vec{AN} \leftarrow \vec{0}$ ;
9:        $AD \leftarrow 0$ ;
10:      for all  $T$  and adjoining triangles  $f_i$  of  $T$  do
11:        Project  $P_H$  against  $f_i$  to obtain  $P_i$ ;
12:        Compute 2D texcoords  $(u_i, v_i)$  of  $P_i$  on  $f_i$ ;
13:        Sample HRM pair  $[\vec{N}_i(u_i, v_i), H_i(u_i, v_i)]$ ;
14:        Evaluate weights  $\omega_i$  from  $P, P_i$  and  $\rho$ ;
15:         $\vec{AN} \leftarrow \vec{AN} + \omega_i \vec{N}_i(u_i, v_i)$ ;
16:         $AD \leftarrow AD + \omega_i H_i(u_i, v_i)$ ;
17:      end for
18:       $\vec{AN} \leftarrow \vec{AN} / |\vec{AN}|$ ;
19:       $AD \leftarrow AD / \sum \omega_i$ ;
20:    else           ▷ Collision against a single face
21:      Sample HRM pair  $[\vec{N}(s, t), H(s, t)]$ ;
22:       $\vec{AN} \leftarrow \vec{N}(s, t)$ ;
23:       $AD \leftarrow H(s, t)$ ;
24:    end if
25:    if ( $\text{penetration} = AD - |\vec{P}_H, \vec{P}| > 0$ ) then           ▷ Positive penetration, a real collision
26:      Calculate force magnitude  $F(\text{penetration})$ ;
27:      Apply  $F$  in the normal  $\vec{AN}$  at the device;
28:    end if
29:  end if
30: end loop

```

---

using a multi-texturing approach. In Figure 4.6(a) we see a schematic of this heightfield stitching. A parametric band of size  $\rho$  extends at both sides of each edge. In this area we use an alpha-blending function. We extend each parametric distance of the triangle's barycentric coordinates in this quantity  $\rho$ , say 0.05 (or 5%) over each HRM. Each blending map of Figure 4.6(b) is then used to compute an averaged mesostructure that spans parametrically a  $\rho$  distance across each edge.

If the projected point of the haptic probe is inside the  $\rho$  band of triangle A (in Figure 4.6(a)), at least one of the barycentric coordinates of  $P_0$  is less than  $\rho$  at some edge. This point is remapped into the opposing triangle B, obtaining its local set of barycentric coordinates  $P_1$ . To blend both heights and normals, we solve for a  $t$  value between 0 and 1 out of  $P_0, P_1$  and  $\rho$  for the chosen blending function, and obtain the corresponding weights  $\omega(t)$ . Several blending functions may be defined for different effects. If we desire no blending at all, a half white/half black map (Figure 4.6(b), No blend) will produce the abrupt relief transition at the edges, generating jumps at edge crossings. A linear gradation from white to black (Figure 4.6(b), Linear) or a sloping S-shaped curve (Figure 4.6(b), S-



**Figure 4.6** Placing per-face HRM textures (heightfield and normalmaps) on top of triangles. To avoid abrupt surface changes crossing surfaces, a blending function is applied around a  $(\rho)$  band along edges. Interpolation can be either linear (i), smoothed (ii) or none (iii).

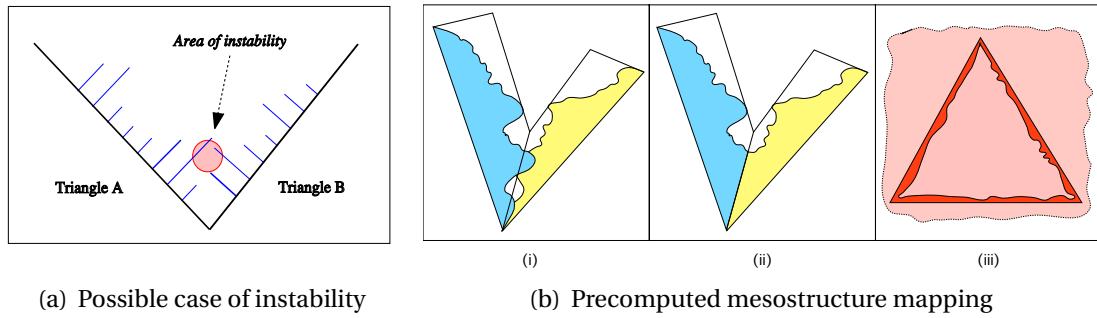
Shape) offer more stable and pleasant results. We use the  $\omega$  weights to compute an average height and normal direction. In the case of point  $P_2$ , some additional barycentric coordinate is also less than  $\rho$ , so this process is repeated for this adjacent edge. Point  $P_3$  falls outside of the  $\rho$  bands, so it is sampled only once.

### 4.3.2 Haptic rendering in concave faces

A problem mentioned before is a performance issue arising when rendering non-convex objects (see Figure 4.7(a)). When two or more triangles form a concave fold or depression (angle between faces less than  $180^\circ$ ) the haptic probe could be inside two (or more) prisms at the same time.

Basically, the treatment of concave faces is the same applied to flat and convex faces. It will blend heights and normals in a band of size  $2\rho$  around the edges (half in one face, half on the other), so the effect will be a repulsion away from the edge. Unfortunately, this may create a back-and-forth effect at the probe (see Figure 4.7(a)), sometimes getting stuck and unable to leave the surface, or in rare cases, generating a resonance situation with ever increasing force magnitude, generating a device failure. Our solution is to transform the initial mesostructure texture, so that height and cumulative normals are already mapped for those surface points that collide into other faces, saving the inclusion and collision tests altogether. In Figure 4.7(b)(i) we see two adjoining triangles and their corresponding unblended mesostructures (blue and yellow), with a subsurface hole (and potential device trap) laying in the middle. At Figure 4.7(b)(ii) we see the result of the blended joint mesostructure, with the hole eliminated. The probe will be pushed away in the combined correct direction.

To avoid borderline cases, we add a small  $\epsilon$  to the collided heights to avoid getting trapped in a narrow crevice or hole. The probe is detected inside the common region by a simple inclusion



**Figure 4.7** *Haptic rendering in concave faces.*

test, and the relevant heightfields (red) reflect the correct surface relief close to the edges (Figure 4.7(b)(iii)).

Stitching different mesostructures may be applied at triangle boundaries if so desired. When the HRMs are precomputed from existing fine geometric detail, abrupt height and normal differences are greatly minimized across edges, thus greatly reducing haptic artifacts.

## 4.4 Testing Procedure

To measure quantitatively the participants' abilities to perceive each surface's haptic properties, we devised the following protocol for choice meshes and textures, using both force shading and HRMs.

### 4.4.1 Setup

A total of 4 (Tests I, II, III and IV) separate experiments (see Table 4.3) were performed on the participants, plus a previous baseline perception test (Test 0) as control setup, so users would recognize what a feature-less surface feels like.

#### Equipment:

A FCS HAPTICMaster, able to exert forces from a delicate 0.01 N up to a heavy blow of 250 N, with a built-in 3D haptic perception wedge of 40 cm x 36 cm x 1 radian. The PC is a 2.4 GHz Pentium IV with an ATI Radeon X1600 graphics card.

#### Stimuli:

We prepared a set of base meshes, shown on Table 4.1, each having some measurable perception property. To create the set of HRM coats shown on Table 4.2, we generated the appropriate height-field displacement maps, and calculated their corresponding normal maps as explained in [120],

(also used in force shading). In the latter manner we could dress any mesh with chosen HRM coats representing particular 3D mesostructures.

**Table 4.1** *Trial model meshes.*

<i>Mesh</i>	<i>Description</i>
$M_a$	Open regular mesh (flat triangle surface)
$M_b$	Softly convex mesh, folding angles slightly over $180^\circ$
$M_c$	Convex mesh, faces folding at acute, square and obtuse angles
$M_{d,j}$	Closed convex meshes (spheres), recursively derived from an initial Icosahedron
$M_e$	Open concave mesh in the shape of a “cup”
$M_f$	Open mesh of a regular gradation of triangles, from big to small
$M_g$	A much denser mesh based on $M_b$ , with very small triangles following the heightfield

### Participants:

Each test involved 18 different user tester/trials (6 participants, 3 trials each) for each setup. Testers were isolated and unaware of what to expect. All of them had used the haptic device before, and were instructed to maintain constant force and speed throughout each experiment. Their perception impressions were recorded from the same live questionnaire.

**Table 4.2** *Hybrid rugosity mesostructures.*

<i>HRM</i>	<i>HRM feature description</i>
$H_0, N_0$	Zero height, constant orthogonal normals texture
$H_{1,k}, N_{1,k}$	Family of serrated textures (vertical left side; sloping right side), with peak frequency growing in $k$
$H_2, N_2$	Raised beams crossing at right angles
$H_3, N_3$	Gently sloping rings, peaks and holes
$H_4, N_4$	Bumps and warts of varying sizes and densities
$H_5, N_5$	Raised flat cylinders (such as a coin)
$H_6, N_6$	Engraved letter S (a narrow groove or scratch forming a <i>purely negative</i> heightfield)
$H_7, N_7$	An irregular fractal landscape

### Procedure:

Each trial consisted in a different  $\langle \text{Mesh}, \text{HRM}, \text{Test} \rangle$  triad being executed. The control experiment (Test 0) was established to rule out perception differences between the geometric and HRM-based renderings of the same meshes.

The experiments (shown on Table 4.3) were performed and measured modulating the maximum amplitude of heightfield displacements, at 1%, 5%, 10%, 15% and 20% of the average edge length of

each mesh. This proved a better predictor than average triangle mesh area, since it works well even with near degenerate triangles.

**Table 4.3** *Trial tests protocol.*

<i>Test Description</i>	<i>Mesh</i>	<i>HRM</i>	<i>What is measured</i>
<b>Test 0.-</b> Baseline perception (Control)	All	$H_0$	<i>Differences between geometric and HRM renderings of same mesh</i>
<b>Test I.-</b> Quality of Visual-Haptic perception	$M_a, M_{d,j}$	$H_2, H_3, H_4$	<i>Perception differences between haptic rendering algorithms</i>
<b>Test II.-</b> Perception quality of monotonous mesostructure (simple patterns)	$M_a, M_b$	$H_{1,k}$	<i>Visual-haptic resolution calibration, height variation, groove counting and orientation</i>
<b>Test III.-</b> Perception quality of non-monotonous mesostructure (complex patterns)	$M_b$	$H_2, H_3, H_4, H_5, H_6, H_7$	<i>Visual-haptic correspondence, height variation, contour following, bumpy quality</i>
<b>Test IV.-</b> Perception of visual-haptic disparity in a gradated mesh	$M_f$	$H_{1,k}, H_2, H_7$	<i>Limits of feature perception at changing resolutions</i>

#### 4.4.2 Test 0 – Baseline perception (Control)

The baseline perception was designed to test whether users would find any differences between a simple collision with the underlying mesh geometry, and the same mesh having its triangles replaced by the flat mesostructure  $[H_0, N_0]$  (of constant zero height throughout).

##### 4.4.2.1 Evaluation of results from Test 0

With respect to mesh  $M_a$  (flat mesh of equal triangles) and  $M_f$  (flat mesh of unequal triangles) all testers detected no edge crossings or features whatsoever. Working with meshes  $M_b, M_c, M_d$  and  $M_e$ , edge crossings were detected only at faces joining at non-flat angles in both approaches. As expected, there was no spurious detection of any other surface feature in both renderings. A similar result was reported for Mesh  $M_g$ , a very dense mesh of minute triangles, which was felt as a continuous surface of varying detail, and beyond the users' ability to detect any edge crossings in the geometry.

#### 4.4.3 Test I – Quality of Visual-Haptic perception

A series of spheres were built recursively at several resolutions out of a regular icosahedron, and then the synthetic HRMs shown on Figure 4.4 were generated. The chosen textures were: alternating polished and variable bumpy areas ( $N_4$  and  $H_4$ ), gently sloping circles within a soft gradient ( $N_3$  and  $H_3$ ), and an embossed cross having only horizontal and vertical surfaces ( $N_2$  and  $H_2$ ).



#### 4.4.3.1 Evaluation of results from Test I

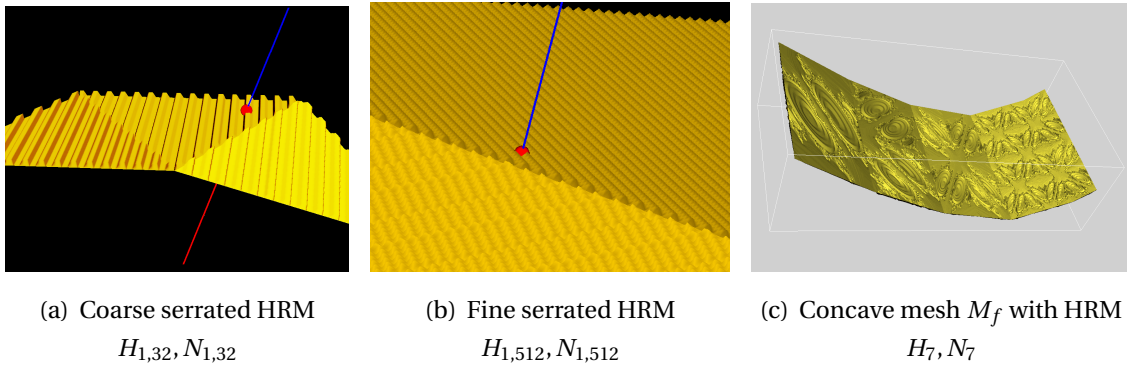
We show on Figure 4.5, different renderings using force shading and HRMs. It is evident from the figures that normals in bump mapping/force shading make for a smoother visualization. However, in terms of haptic perception, the comparisons are quite different and depend on the characteristics of the texture map:

- In the case of the displacement map shown in Figure 4.5(c), using a texture image with a big oval and two small bumps over the surface, the resulting perception is a bit different between the two methods. In the part of small bumps there is almost no difference, but in the big oval part, the force shading method only perceives the resistance for going up to the oval, while in the heightfield method the perception is clearly going up and down from it.
- In the case of the texture shown in Figure 4.5(e) of a cross relief over the surface, the perception is clearly different between the two methods. In the force shading method users perceive resistance on the going up and a jump going down, but report no measurable height differences. In the HRM method the perception is more accurate, because going up and down the cross gives the feeling of real height displacements from the base.

As a summary for this test, we can conclude that our HRM algorithm gives a more accurate sense of surface characteristics than the use of force shading.

#### 4.4.4 Test II – Perception of mesostructure with simple repeating patterns

This test was devised to detect the lower and upper limits of haptic modeling and perception using the HRM approach. The test measured several perception variables out of a regular serrated pattern: How does it feel when going back-and-forth? Can the ridges be counted? Are the height differences noticeable? Each trial was performed with base mesh  $M_b$  using HRMs  $H_{1,j}$ , having regular serrated patterns at different frequencies (with corresponding normals  $N_{1,j}$ , see figures 4.8(a) and 4.8(b)).



**Figure 4.8** Perception scaling adjustment for mesostructure.



For each trial, the maximum heightfield value (that is, the altitude of the prism) was modulated at 1%, 5%, 10%, 15% and 20% of the average edge length, and repeated runs with several users. The *force shading* approach failed this test utterly, detecting just non-directional vibration at higher frequencies, and shown at best to be unreliable at lower ones.

#### 4.4.4.1 Evaluation of results from Test II

When we tested the HRMs, ranging the surface frequencies from few ridges to many, only the last two showed a performance threshold. *Frec256* is a mesostructure that has an asymmetric serrated peak-valley combination repeated 256 times, and *Frec512* is correspondingly doubled. Results are summarized in Figure 4.9(a) and Figure 4.9(b). The red (ridge count) and blue (left-to-right difference) lines in each graph represent the sample mean, and the surrounding shaded areas represent two standard deviations around the mean. Two important facts that can be extracted from this results:

- There exists a definite region for optimum perception of haptic features, having peaks and valleys of 5%-15% of a triangle's edge size, with a "sweet spot" at altitude 10%. The 5%-15% region also holds for dynamic characteristics such as groove sense, and left-to-right or right-to-left differences. At the highest texture resolution, all test subjects only reported directionless vibration.
- Heights greater than 20% produce instabilities in the haptic device, due to fast change in surface normals, high forces in steep vertical walls, and overshoot due to feedback kick.

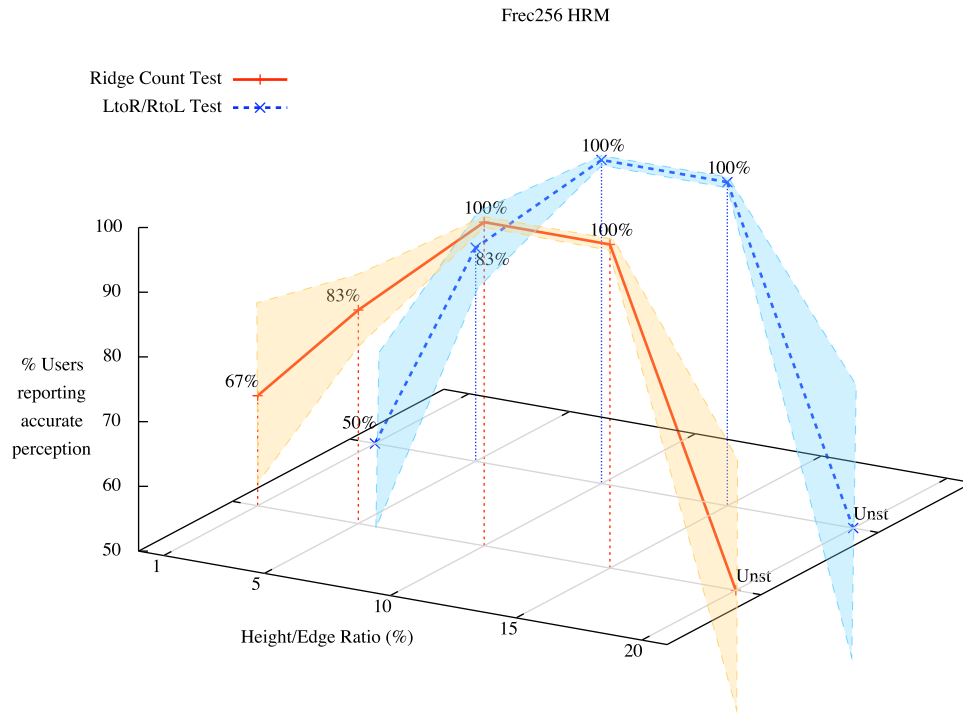
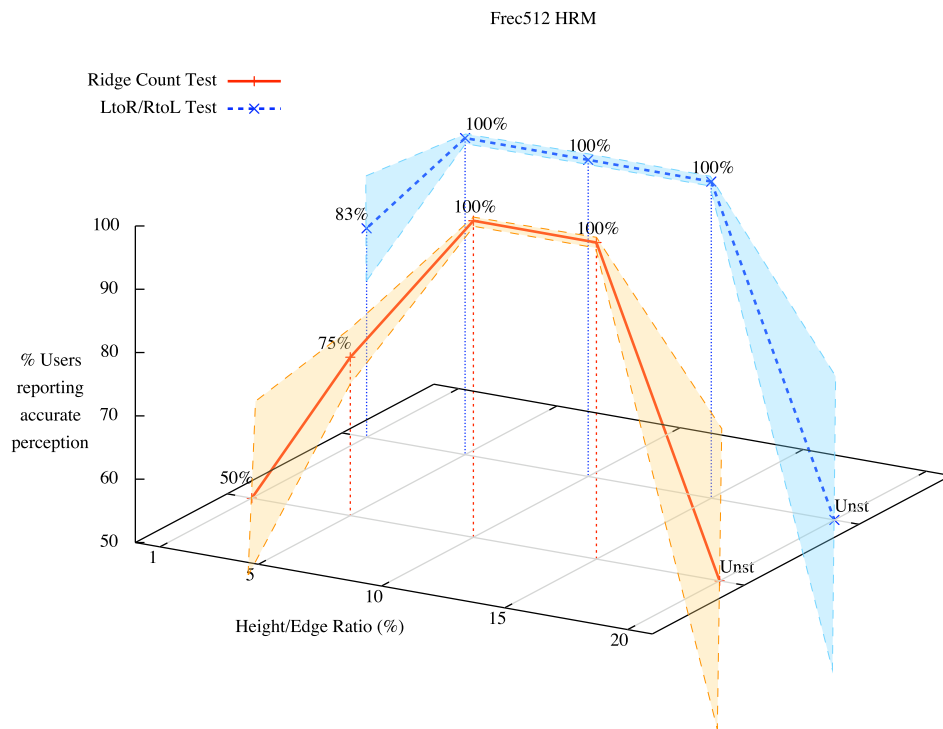
These results hint at a practical threshold on how well mesostructure may be modeled by this approach. In the upper end of the scale, mesh zones whose surface variation exceeds 15% of average edge size are candidates for finer remeshing. In the other end of the scale, if a triangle is perceived as too smooth, the haptic sensation may be enhanced by a coarser sampling of the same texture.

#### 4.4.5 Test III – Perception of non-monotonous mesostructure

Here we measured the ability to perceive definite shapes in the haptic textures: a soft texture of sloping peaks and depressions; small-to-big warts; single scratches. The object of this test is the multi-modal quality of perception: how it corresponds with the visual representation and whether it can be "followed along".

##### 4.4.5.1 Evaluation of results from Test III

As can be extracted from Table 4.4, even small scratches are felt and followed. All testers were able to accurately detect the target features even at low resolutions, except when reaching the 20% threshold level, at which point instability set in and perception degraded quickly.

(a) Test results for HRM  $H_{1,256}, N_{1,256}$ (b) Test results for HRM  $H_{1,512}, N_{1,512}$ **Figure 4.9** Haptic perception of surface frequencies at medium & high resolutions.

**Table 4.4** *Haptic perception of fine features in non-monotonous mesostructure*

Test	1%	5%	10%	15%	20%
<i>Straight walls</i>	100% yes	100% yes	100% yes	100% yes	unstable
<i>Round contours</i>	73% yes	100% yes	100% yes	100% yes	unstable
<i>Light scratches</i>	100% yes	100% yes	100% yes	100% yes	unstable
<i>Soft slopes</i>	100% yes	100% yes	100% yes	100% yes	unstable
<i>Small bumps</i>	100% yes	100% yes	100% yes	83% yes	unstable

#### 4.4.6 Test IV – Perception of visual-haptic disparity in a gradated mesh

We measure resolution changes in perception. We map the same haptic texture into a mesh ( $M_f$ ) made of rectangular triangles of decreasing size, in order to test the limits of perception, aliasing effects and arising instabilities. We also measure how these qualities change as we zoom (both haptically and visually) in the mesh.

##### 4.4.6.1 Evaluation of results from Test IV

In trial mesh  $M_f$  (Figure 4.8(c)), neighboring triangles progressively reduce their area in half from left to right (height is reduced by  $\sqrt{2}/2$ ). Since mesostructure remains at the same resolution, the resulting mapped areas actually double their density from left to right, and sampling aliasing occurs. What is obtained is a sequence of similar shapes, decreasing in perceptible features.

Acute features noticeable in big triangles become fainter at smaller triangles. If the scene is zoomed in (or out) they become sharper (or smoother) again. A feature becomes undetectable when the height difference becomes less than a corresponding visual pixel, just as expected by the Nyquist limit. In other words, if a visual difference is seen, then it should be felt. This is completely in accord with user's expectancies.

An important result is the existence of a definite region for optimal perception of haptic features, with textures having maximum peak heights or valley depths between 5%-15% of an average edge size, which also holds for dynamic characteristics such as groove sense, and left-to-right or right-to-left traversal differences. This includes a blending procedure across edges that smoothes surface traversals even when transitioning between two or more HRM textures

A further result from this test is that the constant speed and very low complexity of computations needed for collision detection, HRM texture sampling, and haptic force rendering actually allow simple models to feel as complex as much denser geometric meshes. The haptic sampling rates (200 Hz to 1000 Hz) measured for the subject meshes in our approach are faster than those reported in [69] for comparable geometric models.

## 4.5 Conclusions from this chapter

We developed a fast and accurate method for rendering per-face local haptic texture in triangle meshes, allowing perception of correct surface details at several resolutions, and extending the use of heightfield haptics to perceive higher surface detail without modeling it geometrically. The proposed method allows rendering of HRM-textured triangle meshes to feel as detailed as very dense geometric meshes, at a fraction of the modeling and processing costs. The sampling's constant (and very low) computation speed accounts for faster collision detection than those reported in the literature, and promises to scale well even for huge models.

Our HRM-rendering algorithm allows accurate perception and traversal along the grooves of scratches, as generated in [10]. In those cases, *force shading* does not give a correct perception because neighboring points with converging normals actually push the haptic probe away from the scratch. It shows ample suitability for modeling and perceiving (in real-time) very complex surface textures of varying frequency, which can be layered on top of simpler geometric models such as bones, major body organs, machine assembly pieces and other structures.

A result of great value is the creation of a repository of base meshes and standard HRM textures, allowing measuring quantitative and qualitative differences among the haptic rendering algorithms described herein and others that might follow, using the battery of tests devised for this research.

In order to apply our method for perceiving overlaid scratches on surfaces [10], we have extended it to accept HyRMAs representing inverse heightfields. In those cases, *force shading* does not give a correct perception because neighboring points with converging normals actually push the haptic probe away from the scratch. Our HyRMA-rendering algorithm allows accurate perception and traversal along the grooves of the scratches, and shows ample suitability for modeling and perceiving (in real-time) very complex surface textures of varying frequency, which can be layered on top of simpler geometric models such as bones, major body organs, machine assembly pieces and other structures.

Our model allows adding material friction as a constant global coefficient, or expanding the HyRMA with a second 2D texture field whose value represents a variable friction coefficient at each triangle point. This may include incorporating the added resistance of fine microstructure surface properties into the model. We expect to measure favorable performance differences if using an HyRMA-based approach instead of a pure geometric model in rendering haptic collisions, and thus obtain a robust result for the method.

Two problems have to be solved in order to obtain a global solution for this approach to work. The first is finding a way of representing surface relief in a continuous mesostructure encoding that uses prisms volumes but does away with separate local textures, and avoiding completely the stitching problem at the edges, by exploring a procedure to scan the fine triangle geometry of dense meshes, and replace it with a decimated mesh of larger triangles while capturing most of the perceptible frequency details of the original surface in a blended global HRM mesostructure atlas (height

displacements, surface normals and other properties such as directed friction and stickiness). The second one is adding a suitable haptic collision scheme adjusted to the former method. These will be the topics considered in the next two chapters.

Results from this research have been published in [\[138\]](#), [\[140\]](#) and [\[139\]](#).

# Collision detection in conformal space



... a time of tragic collision between matter and spirit  
and of the downfall of the purely material world view.

Wassily Kandinsky

Certain application domains, such as animation and haptic rendering, involve real-time interactions among detailed models, requiring fast computation of massive numbers of collisions. Diverse formulations and optimizations have been developed that individually target specific object representations for collision queries.

In the next sections we present an unified geometric algebra treatment that shifts collision detection from euclidean  $\mathbb{R}^3$  space to higher  $\mathbb{R}^{4,1}$  conformal space, providing an elegant abstraction to compute collisions among segments, areas, spheres, and polyhedral objects, treating them as similar conformal primitive entities. A SIMD parallel implementation of CUDA kernels running in GPU was devised to provide performance improvements due to the fourfold increase in dimensionality.

Results show expected interactive rates improvements when computing collisions and intrusions among known mesh models, and just below the optimal rates for a standard CPU implementation, even when not including hierarchical collision pre-filtering schemes.

## 5.1 Geometric Algebra

A recent formalism in Computer Graphics is the Geometric Algebra approach described by Dorst *et al* [30], with the extensions to conformal geometric spaces added by Vince [147]. This algebra has three fundamental operators: the known inner (*dot*) product ( $x \cdot y$ ), the outer (*wedge*) product ( $x \wedge y$ ) and the geometric product ( $xy$ ). As a vector space, it shares other properties of a vector algebra, such as Euclidean distance, invariance, etc.

**Definition 5.1.1** For vectors  $a$  and  $b$ , the outer product  $a \wedge b$  defines an oriented hyperplane, or bivector as defined by Hestenes [50]. Its magnitude is the signed area of the parallelogram  $\|a \wedge b\| = \|a\|\|b\| \sin\theta$ . The orientation of the signed area (the resulting parallelogram) is said to be positive when  $a$  folds onto  $b$  counterclockwise, and negative otherwise.

**Definition 5.1.2** For vectors  $a$  and  $b$ , its geometric product  $ab$  is the sum of the inner (*dot*) product and the outer (*bivector*) product, with the following properties:

$$ab = a \cdot b + a \wedge b, \quad \begin{cases} \text{antisymmetric,} & ba = a \cdot b - a \wedge b \\ \text{associative,} & a(bc) = (ab)c = abc \\ \text{distributive,} & a(b+c) = abd + acd \end{cases}$$

From these definitions are derived the following algebraic properties

$$\begin{aligned}
 a \wedge a &= 0 & a^2 &= a \cdot a = \|a\|^2 \\
 (\lambda a) \wedge b &= \lambda(a \wedge b), \text{ for scalar } \lambda & (\lambda a)b &= \lambda(ab), \text{ for scalar } \lambda \\
 b \wedge a &= -a \wedge b = a \wedge (-b) & a \wedge (b + c) &= a \wedge b + a \wedge c
 \end{aligned}$$

### 5.1.1 Conformal Geometry

Conformal Geometry [8, 29, 30] describes an elegant algebraic space for geometric visualization in  $\mathbb{R}^3$ , since it is homogeneous, supports points and lines at infinity, preserves angle and distance, and can represent points, circles, lines, spheres and planes.

**Definition 5.1.3** A conformal space  $\mathbb{R}^{p+1,q+1}$  of  $p+1$  positive dimensions and  $q+1$  negative dimensions is built from a  $\mathbb{R}^{p,q}$  space. A point  $x = ue_1 + ve_2 + we_3$  in  $\mathbb{R}^3 (\equiv \mathbb{R}^{3,0})$  maps to a null vector  $X$  in  $\mathbb{R}^{4,1}$  ( $X \cdot X = 0, X \neq 0$ ), having the orthonormal base  $\{e_1, e_2, e_3, e, \bar{e}\}$ .

$$\begin{aligned}
 e_1 \cdot e_1 &= 1, & e_2 \cdot e_2 &= 1, & e_3 \cdot e_3 &= 1 \\
 e \cdot e &= 1, & \bar{e} \cdot \bar{e} &= -1 \\
 n_\infty &= e + \bar{e}, & n_o &= \frac{1}{2}(e - \bar{e}) \\
 X = P(x) &= x + \frac{1}{2}x^2 n_\infty + n_o
 \end{aligned} \tag{5.1.1}$$

with  $n_\infty$  and  $n_o$  representing the null vectors at infinity and at the origin respectively.

**Definition 5.1.4** The outer product of  $k$  vectors is called a  $k$ -blade:

$$v_1 \wedge v_2 \wedge \dots \wedge v_{k-1} \wedge v_k = V \in \mathbb{R}^{p,q}, \quad k \leq p+q \tag{5.1.2}$$

The highest order  $k$ -blade of a  $\mathbb{R}^{p,q}$  space is called a pseudoscalar, denoted by  $I$  ( $I^2 = -1$ ), for its similarity as a rotor to the complex number  $\mathbf{i}$ . Thus,  $I X$  is a  $\frac{\pi}{2}$  counterclockwise rotation of  $X$ . Likewise,  $X I$  is a  $\frac{\pi}{2}$  clockwise rotation of  $X$ .

Table 5.1 shows the 32 blade terms of the canonical base in  $\mathbb{R}^{4,1}$ . A *multivector* is a linear combination of the  $\sum_{k=1}^{p+q} \binom{p+q}{k} = 2^{p+q}$  canonical base of possible blades for the conformal  $\mathbb{R}^{p,q}$  space.

If we write  $e_i e_j \dots e_k = e_{ij\dots k}$ , then

$$M = \lambda_0 + \lambda_1 e_1 + \lambda_2 e_2 + \lambda_3 e_3 + \lambda_{12} e_{12} + \lambda_{23} e_{23} + \lambda_{31} e_{31} + \lambda_{123} e_{123} \quad (\text{for } \mathbb{R}^3) \tag{5.1.3}$$

From these null vectors are derived the primitives shown on Table 5.2.

### 5.1.2 Intersections in conformal space

Many modeling and transformation problems are solved within euclidean spaces. However, in other spaces some operations may be simpler to compute. A well known example are 4D projective spaces

**Table 5.1** The 32 (1+5+10+10+5+1) terms of the canonical base for the conformal  $\mathbb{R}^{4,1}$  space.

Base Elements	Blade components				
1 scalar	$\lambda$				
5 vectors	$e_1$	$e_2$	$e_3$	$e$	$\bar{e}$
10 bivectors	$e_1 \wedge e_2$ $e_3 \wedge \bar{e}$	$e_2 \wedge e_3$ $e_1 \wedge e$	$e_3 \wedge e_1$ $e_2 \wedge e$	$e_1 \wedge \bar{e}$ $e_3 \wedge e$	$e_2 \wedge \bar{e}$ $\bar{e} \wedge e$
10 trivectors	$e_1 \wedge e_2 \wedge e_3$ $e_2 \wedge e_3 \wedge \bar{e}$	$e_1 \wedge e_2 \wedge \bar{e}$ $e_2 \wedge e_3 \wedge e$	$e_1 \wedge e_2 \wedge e$ $e_1 \wedge \bar{e} \wedge e$	$e_3 \wedge e_1 \wedge \bar{e}$ $e_2 \wedge \bar{e} \wedge e$	$e_3 \wedge e_1 \wedge e$ $e_3 \wedge \bar{e} \wedge e$
5 quadvectors	$e_1 \wedge e_2 \wedge e_3 \wedge \bar{e}$	$e_1 \wedge e_2 \wedge e_3 \wedge e$	$e_1 \wedge e_2 \wedge \bar{e} \wedge e$	$e_3 \wedge e_1 \wedge \bar{e} \wedge e$	$e_2 \wedge e_3 \wedge \bar{e} \wedge e$
1 pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge \bar{e} \wedge e$ (I)				

[146] in homogeneous coordinates that provide an unique description for translations, rotations, scaling and perspective projections simplifying otherwise complex computations in 3D space.

In this vein, conformal geometry [8] describes an alternative space within the framework of geometric algebra, whose characteristics offer an elegant alternative for solving geometric and visualization problems in  $\mathbb{R}^3$ , being some of them:

- Is homogeneous.
- Supports points and lines at infinity.
- Simple unified representation of geometric objects such as points, lines, planes, circles, spheres, etc.
- Preserves angles and distances.

The *meet* operator ( $\vee$ ) denotes the *intersection multivector*, having pseudoscalar  $I = e_1 e_2 e_3 e \bar{e}$  for that space. Intersections among multivectors in the conformal model [29] are specified by the same equation for all multivectors, as shown in Table 5.3.

$$B = (X \vee Y) = (I X) \cdot Y, \text{ with square norm } B^2 = \|B\|^2 \quad (5.1.4)$$

$$\text{and } B = \beta_0 + \beta_{e_1} e_1 + \dots + \beta_{e_1 e_2} e_1 e_2 + \dots + \beta_{e_1 e_2 e_3 e \bar{e}} e_1 e_2 e_3 e \bar{e}.$$

in which  $(X \vee Y)$  reads as the geometric product of the pseudoscalar  $I$  and multivector  $X$ , and then dot-multiplied by multivector  $Y$ . This operator is mostly used for collision detection in computer graphics. Its algebraic fundamentals can be found in the works by Doran & Lasenby [29] and Dorst,



**Table 5.2** Algebraic primitives built from blades in the  $\mathbb{R}^{4,1}$  conformal space, which also includes scalars, points and vectors

Primitive	Blade type	Algebraic representation	Geometric interpretation
Circle	<i>trivector</i>	$C = P_1 \wedge P_2 \wedge P_3$	Three noncollinear points delimit the perimeter of the circle.
Line	<i>trivector</i>	$L = P_1 \wedge P_2 \wedge n_\infty$	Two nonidentical points define a segment plus the point at <i>infinity</i> .
Sphere	<i>quadvactor</i>	$S = P_1 \wedge P_2 \wedge P_3 \wedge P_4$	Four noncoplanar points delimit the surface of the sphere.
Plane	<i>quadvactor</i>	$\Pi = P_1 \wedge P_2 \wedge P_3 \wedge n_\infty$	Three noncollinear points define a triangle plus the point at <i>infinity</i> .
Area	<i>bivector</i>	$A = q_1 \wedge q_2$	Two nonparallel vectors forming a parallelogram.
Volume	<i>trivector</i>	$V = q_1 \wedge q_2 \wedge q_3$	Three nonparallel vectors forming a parallelepiped.
Pseudoscalar	<i>5-vector</i>	$I = e_1 \wedge e_2 \wedge e_3 \wedge e \wedge \bar{e}$	The pseudoscalar ( <i>rotor</i> ) for the $\mathbb{R}^{4,1}$ conformal space.

**Table 5.3** Primitive intersections in  $\mathbb{R}^{4,1}$  conformal space

Primitive	Primitive	Conformal representation
Line	Plane	$B = \Pi_1 \vee L_1 = (I \Pi_1) \cdot L_1$
Line	Sphere	$B = S_1 \vee L_1 = (I S_1) \cdot L_1$
Plane	Plane	$B = \Pi_1 \vee \Pi_2 = (I \Pi_1) \cdot \Pi_2$
Plane	Sphere	$B = S_1 \vee \Pi_1 = (I S_1) \cdot \Pi_1$
Sphere	Sphere	$B = S_1 \vee S_2 = (I S_1) \cdot S_2$

Fontijne & Mann [30]. The work of Roa [110] states the following criteria for  $B^2$ :

$$if \begin{cases} B^2 > 0, & \text{intersection at least at two points} \\ B^2 = 0, & \text{intersection at a tangent point} \\ B^2 < 0, & \text{primitives do not intersect} \end{cases} \quad (5.1.5)$$

## 5.2 Kernels for intersection algorithms

For the next sections we present the algorithms, results and conclusions of the CUDA implementation of kernel code fragments for collision detection, using meshes from the Stanford University repository at <http://www.graphics.stanford.edu/data/3Dscanrep/>. Neither bounding volume hierarchies nor optimizations were employed, just raw collisions.

The obtained algorithms of interest are Ray-Plane (Line Segment-Triangle), Plane-Plane (Triangle-

Triangle), and Sphere-Sphere. The different  $B$  multivectors and their  $B^2$  norms were algebraically derived for each CUDA kernel.

### 5.2.1 Line Segment (Ray)–Triangle (Plane) intersection

The intersection between a line segment and a triangle is a collision query between the ray passing along the segment and the plane of the triangle, and later checking whether boundaries meet. The intersection multivector  $B = (\Pi_1 \vee L_1) = (I \Pi_1) \cdot L_1$  evaluates to

$$\begin{aligned} B = & (\omega_2 \beta_3 + \omega_1 \beta_4 - \omega_4 \beta_1) e_1 e + (\omega_2 \beta_3 + \omega_1 \beta_4 - \omega_4 \beta_1) e_1 \bar{e} \\ & + (\omega_3 \beta_1 - \omega_2 \beta_2 + \omega_1 \beta_5) e_2 e + (\omega_3 \beta_1 - \omega_2 \beta_2 - \omega_1 \beta_5) e_2 \bar{e} \\ & + (-\omega_3 \beta_3 + \omega_4 \beta_2 + \omega_1 \beta_6) e_3 e + (-\omega_3 \beta_3 + \omega_4 \beta_2 + \omega_1 \beta_6) e_3 \bar{e} \\ & + (-\omega_3 \beta_4 - \omega_4 \beta_5 - \omega_2 \beta_6) e \bar{e} \end{aligned} \quad (5.2.1)$$

$$B^2 = (\omega_3 \beta_4 + \omega_4 \beta_5 + \omega_2 \beta_6)^2 \quad (5.2.2)$$

where the  $\beta$ s and the  $\omega$ s are the coefficients of the corresponding multivectors for  $L_1$  and  $\Pi_1$ . A nonnegative  $B^2$  signals a potential collision. The segment is then tested against the triangle's edges, to detect crossings and an effective collision. Kernel code fragment 5.1 describes the complete procedure to compute intersections.

### 5.2.2 Triangle (Plane)–Triangle (Plane) intersection

A triangle–triangle is the most interesting collision to define, since is most commonly used.  $B$  is the following term

$$\begin{aligned} B = & (\omega_4 \lambda_2 - \omega_2 \lambda_4) e_1 e \bar{e} + (\omega_2 \lambda_3 - \omega_3 \lambda_2) e_2 e \bar{e} + (\omega_3 \lambda_4 - \omega_4 \lambda_3) e_3 e \bar{e} \\ & + (\omega_2 \lambda_1 - \omega_1 \lambda_2) e_1 e_2 e + (\omega_3 \lambda_1 - \omega_1 \lambda_3) e_2 e_3 e + (\omega_4 \lambda_1 - \omega_1 \lambda_4) e_3 e_1 e \\ & + (\omega_2 \lambda_1 - \omega_1 \lambda_2) e_1 e_2 \bar{e} + (\omega_3 \lambda_1 - \omega_1 \lambda_3) e_2 e_3 \bar{e} + (\omega_4 \lambda_1 - \omega_1 \lambda_4) e_3 e_1 \bar{e} \end{aligned} \quad (5.2.3)$$

$$B^2 = (\omega_4 \lambda_2 - \omega_2 \lambda_4)^2 + (\omega_2 \lambda_3 - \omega_3 \lambda_2)^2 + (\omega_3 \lambda_4 - \omega_4 \lambda_3)^2 \quad (5.2.4)$$

Instead of plane intersections, it is much faster to implement a Triangle–Triangle intersection (see Table 5.3) for three Segment–Triangle intersections, as shown in Code Fragment 5.2. Any one segment colliding with the opposite triangle triggers detection.

**Kernel Fragment 5.1** *Line Segment-Triangle intersection*

```

1 kernel Segment_Triangle_Intersect(segment L1, plane P1)
2   Normalize(L1);   Normalize(P1)
3   [a,e3er] = ConformalIntersectLinePlane(L1,P1)
4   L3 = Line(L0.point1,L0.point2)
5   L2 = Line(P1.point3,P1.point1)
6   [out1,ind1] = ConformalIntersectSegmentSegment(L2, L3 )
7   L2 = Line(P1.point1,P1.point2)
8   [out2,ind2] = ConformalIntersectSegmentSegment(L2, L3 )
9   L2 = Line(P1.point3,P1.point2)
10  [out3,ind3] = ConformalIntersectSegmentSegment(L2, L3 )
11  // out# = 1 (segments intersect); 0 (they do not)
12  // ind#: scalar coefficient of e vector
13  if (a == 0) then //line and plane parallel
14    if (e3er == 0) then //line lies on the triangle's plane
15      //verify segment intersection with other triangles
16      return (out1==1) or (out2==1) or (out3==1)
17    else return 0 // No intersection found
18  end if
19  else //whether both points are in same side of plane
20    sign1 = trivector(P1.point2-P1.point1,P1.point3-P1.point1,L1.point1-P1.point1)
21    sign2 = trivector(P1.point2-P1.point1,P1.point3-P1.point1,L1.point2-P1.point1)
22    if (sign1 == sign2) then
23      return 0 // Segment does not touch plane
24    end if // Segment crosses the plane
25    return result = (ind1>0 and ind2>0 and ind3<0) or
26                  (ind1<0 and ind2<0 and ind3>0);
27  end if

```

**5.2.3 Sphere-Sphere intersection**

$$\begin{aligned}
B = & (\mu_3\lambda_1 - \mu_1\lambda_3)e_1e_2e + (\mu_4\lambda_1 - \mu_1\lambda_4)e_2e_3e + (\mu_5\lambda_1 - \mu_1\lambda_5)e_3e_1e \\
& + (\mu_3\lambda_2 - \mu_2\lambda_3)e_1e_2\bar{e} + (\mu_4\lambda_2 - \mu_2\lambda_4)e_2e_3\bar{e} + (\mu_5\lambda_2 - \mu_2\lambda_5)e_3e_1\bar{e} \\
& + (\mu_5\lambda_3 - \mu_3\lambda_5)e_1e\bar{e} + (\mu_3\lambda_4 - \mu_4\lambda_3)e_2e\bar{e} + (\mu_4\lambda_5 - \mu_5\lambda_4)e_3e\bar{e} \\
& + (\mu_1\lambda_2 - \mu_2\lambda_1)e_1e_2e_3
\end{aligned} \tag{5.2.5}$$

$$\begin{aligned}
B^2 = & -(\mu_1\lambda_2 - \mu_2\lambda_1)^2 - (\mu_3\lambda_1 - \mu_1\lambda_3)^2 - (\mu_4\lambda_1 - \mu_1\lambda_4)^2 - (\mu_5\lambda_1 - \mu_1\lambda_5)^2 \\
& + (\mu_3\lambda_2 - \mu_2\lambda_3)^2 + (\mu_4\lambda_2 - \mu_2\lambda_4)^2 + (\mu_5\lambda_2 - \mu_2\lambda_5)^2 + (\mu_5\lambda_3 - \mu_3\lambda_5)^2 \\
& + (\mu_3\lambda_4 - \mu_4\lambda_3)^2 + (\mu_4\lambda_5 - \mu_5\lambda_4)^2
\end{aligned} \tag{5.2.6}$$

As before,  $B^2$  will determine if a collision occurs. Code fragment 5.3 turned out to be as simple as it is in  $\mathbb{R}^3$ : one of the spheres is placed at the origin and the others moved accordingly. If the distance between the centers is less than the sum of the radii, the spheres intersect, enabling fast computation of huge numbers of colliding

**Kernel Fragment 5.2** *Triangle-Triangle intersection*

```

1 kernel Segment_Plane_Intersect(triangle P1, triangle P2 )
2   //Verify if all points are at same side of plane
3   if not VerifySameSidePoints(P1) then
4       return 0 // No Intersection
5   end if
6   Normalize(P1); Normalize(P2)
7   r1 = Line(P2.point1,P2.point2)
8   r2 = Line(P2.point2,P2.point3)
9   r3 = Line(P2.point3,P2.point1)
10  // out# = 1, intersection exists, 0 no intersection,
11  out1 = ConformalIntersectSegmentPlane(r1, P1 )
12  out2 = ConformalIntersectSegmentPlane(r2, P1 )
13  out3 = ConformalIntersectSegmentPlane(r3, P1 )
14  if (out1 == 1) or (out2 == 1) or (out3 == 1) then
15      return 1 // intersection exists
16  end if
17  r1 = Line(P1.point1,P1.point2)
18  r2 = Line(P1.point2,P1.point3)
19  r3 = Line(P1.point3,P1.point1)
20  out1 = ConformalIntersectSegmentPlane(r1, P2 )
21  out2 = ConformalIntersectSegmentPlane(r2, P2 )
22  out3 = ConformalIntersectSegmentPlane(r3, P2 )
23 return (out1 == 1) or (out2 == 1) or (out3 == 1)

```

**Kernel Fragment 5.3** *Sphere-Sphere intersection*

```

1 kernel Sphere_Sphere_Intersect(sphere S1, sphere S2 )
2   // origin set at the center of sphere S1
3   ChangeCoordinatesSphere(S1)
4   ChangeCoordinatesSphere(S2)
5 return ConformalIntersectSphereSphere(S1,S2) // 1 = spheres intersect, 0 they do not

```

## 5.3 CUDA implementation and results

An initial implementation phase was devised in which the  $\mathbb{R}^3$  to  $\mathbb{R}^{4,1}$  algebraic mappings and algebraic algorithms were prototyped in MatLab™ and linked to AutoDesk Maya™. After checking for algebraic correctness, they were migrated to a CUDA implementation. Trials were performed on a 3 Ghz Dual Core Intel 2 CPU w/ NVIDIA 9800GT 64 core GPU.

GPU performance tests were executed on intersections and collisions in conformal space among several standard meshes to gather statistics. On average, they show a three order of magnitude improvement for all implemented algorithms from a pure (single core) CPU implementation of conformal space.

The basic CUDA procedure allows for querying whether an object, in this case a polygonal mesh, collides against any of the three primitives: line segments, triangles, and spheres. All CUDA kernels share the conformal collision query procedure, with a different post-processing phase.

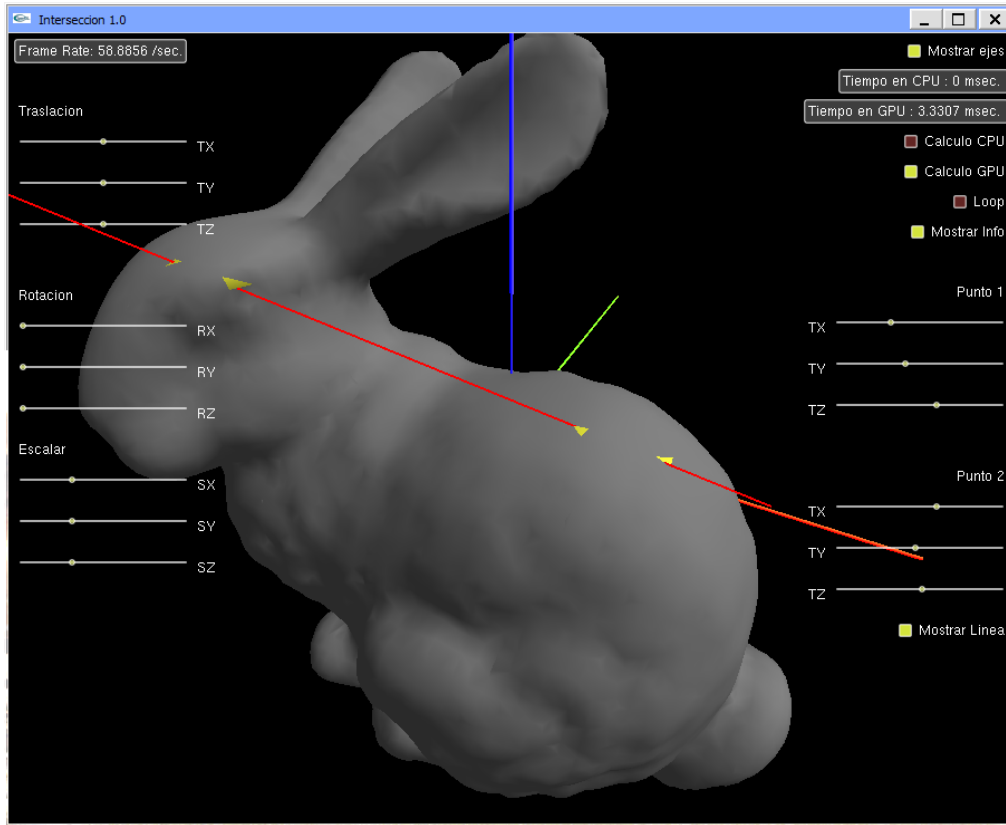
### 5.3.1 Line Segment-Mesh collisions

A line segment (ray) is intersecting 4 triangles from the Stanford Bunny mesh, shown in Figure 5.1. As can be appreciated in Table 5.4, the GPU implementation offers dramatic speedup.

**Table 5.4** CPU 5D – GPU 5D performance evaluation.

Implementation	Triangles	Milliseconds	Seconds
CPU 5D	4 million	30034.40	30.04
GPU 5D	4 million	715.87	0.72

For example, taking the values of the Bunny – Line Segment intersection, when intersecting a line segment with nearly 4 million triangles, GPU computations are still under 1 second.



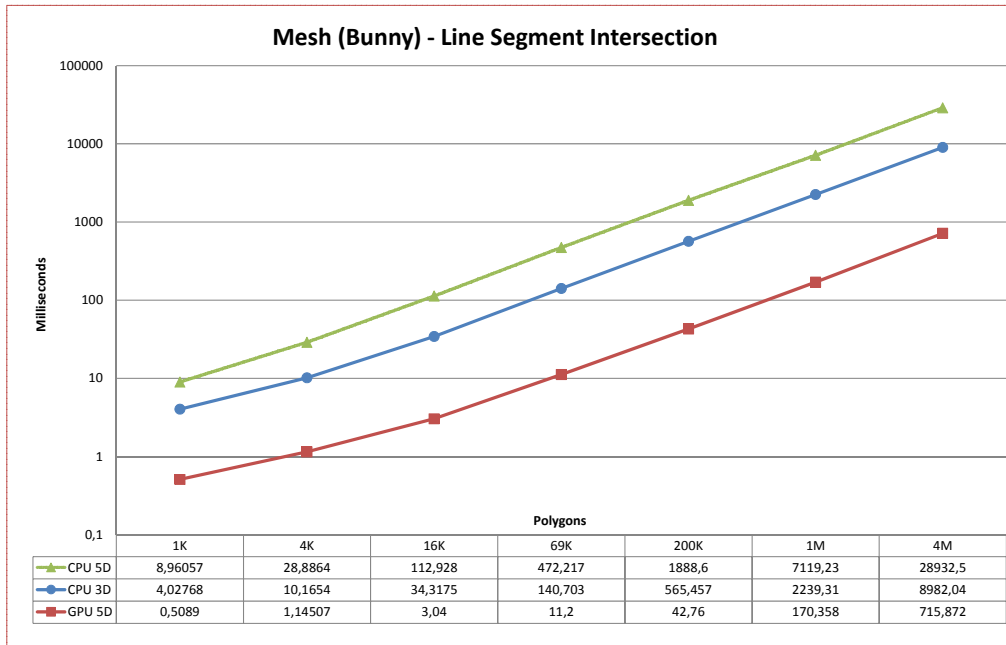
**Figure 5.1** Bunny Mesh – Line Segment Intersection. Colliding polygons shown in yellow.

### 5.3.2 $\mathbb{R}^{4,1}$ (CPU 5D) vs. $\mathbb{R}^3$ (CPU 3D) vs $\mathbb{R}^{4,1}$ (GPU 5D) collisions

For a measuring framework of the conformal approach, Möller's optimized CPU approach for Triangle-Triangle intersection [86] was implemented. The  $\mathbb{R}^{4,1}$  conformal model was also implemented

purely in CPU and a performance evaluation was obtained. We then compare both against our  $\mathbb{R}^{4,1}$  GPU implementation (GPU 5D) to check for time improvements.

We can appreciate in Figure 5.2 that Möller’s CPU 3D implementation (in **blue**) is one order of magnitude faster than the CPU implementation of the conformal model CPU 5D (in **green**), attributed to the extra dimensionality of the latter. Then again, it can be appreciated that our conformal GPU 5D implementation (in **red**) is one order of magnitude *faster* than Möller’s CPU 3D implementation, and two orders of magnitude against a pure CPU 5D implementation, so ours is more efficient in this respect, and its curve grows much slower than the CPU implementation.



**Figure 5.2** *Bunny Mesh – Line Segment Intersection times.*

How would fare the GPU 5D approach against an implementation in GPU of Möller’s 3D collision procedure? We do not have reference values for that implementation, but from these references (Georgii [39], and Shumskiy [121]) we can extract benchmark times for their largest tested polygonal meshes (having hundred of thousands of triangles). Unfortunately, they all use pre-filtering schemes that do not disaggregate or differentiate between time navigating some bounding volume hierarchy and the time computing primitive intersections. It is to be expected that if we implement the same measures our times will drop another order of magnitude. In Table 5.5 we can see values for meshes in the former references, compared against the one we used.

**Table 5.5** GPU 3D collision benchmark times.

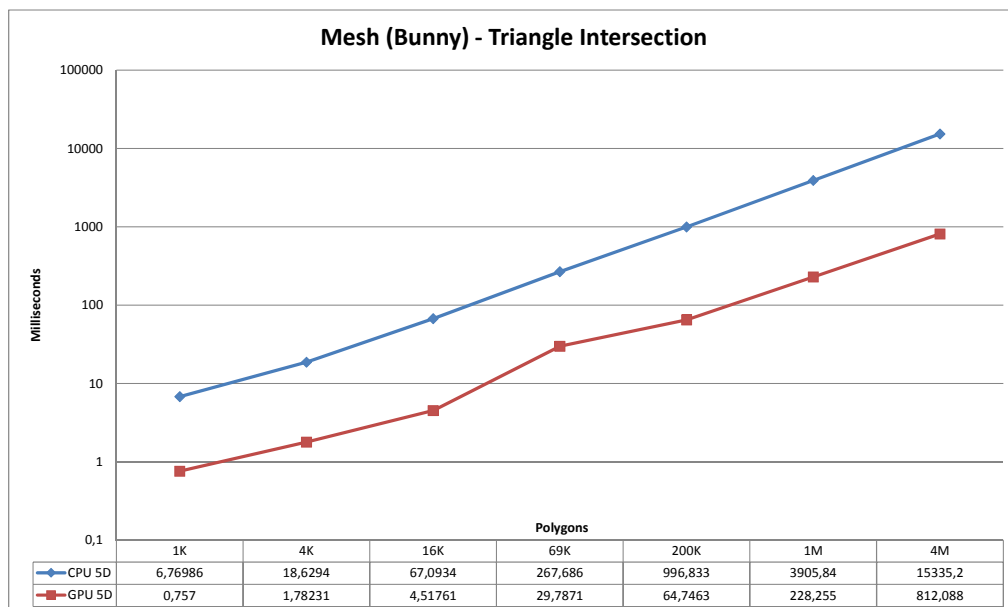
Scheme	Mesh	Triangles	Milliseconds
GPU 5D (ours)	bunny	1000k	228
GPU 3D (Georgii)	bunny	500k	95
GPU 3D (Pabst)	n-body	146k	78

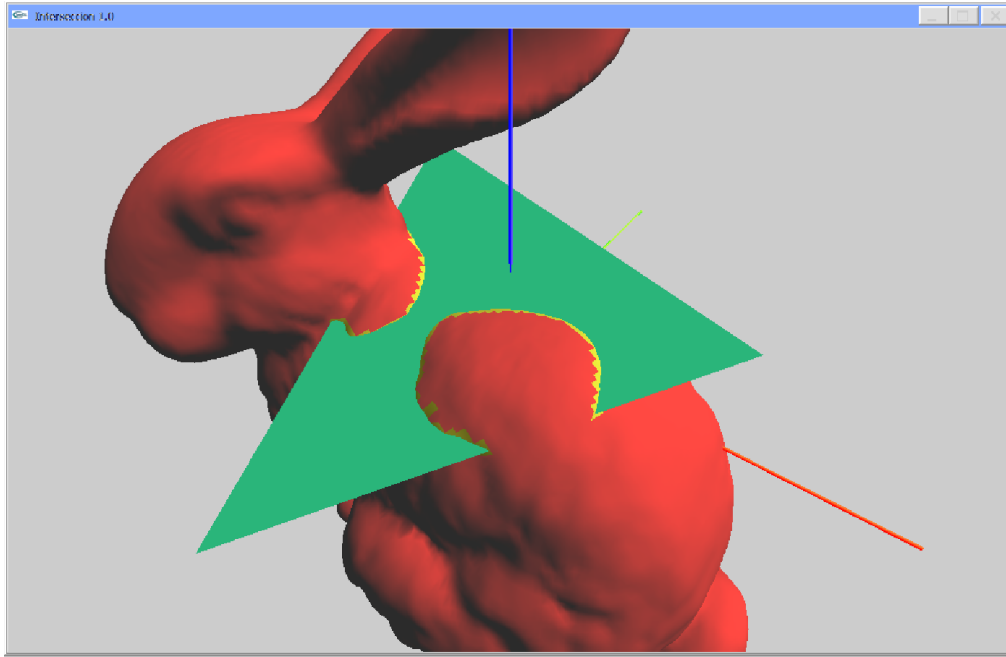
An additional difficulty for accurate comparison is that they did not perform a uniform measurement procedure. We use the same model built at different resolutions, and they rely on different mesh models for each resolution, so geometric variation is affected by dissimilar topologies and level of detail.

The main result here is that our Line Segment–Mesh collision approach in conformal 5D space offers close to optimal framerates for repeated interactive collision computation, and suitable for being incorporated into any haptic rendering algorithm.

### 5.3.3 Mesh-Mesh collisions

A typical computed collision between a large triangle (green) and the Stanford Bunny mesh (red) can be seen in Figure 5.4, with the intersected mesh triangles shaded in yellow to show where the plane (triangle) cuts the mesh.

**Figure 5.3** Fast CPU vs. Conformal CPU Mesh-Triangle Intersections



**Figure 5.4** *Plane (Triangle) – Bunny mesh Intersection: Colliding polygons shown in yellow.*

Here in Table 5.6 are shown the number of collided triangles between the Bunny mesh, kept fixed at 1024 polygons and the Armadillo meshes varying from 5 thousand to 5 million triangles.

**Table 5.6** *GPU 3D collision benchmark times*

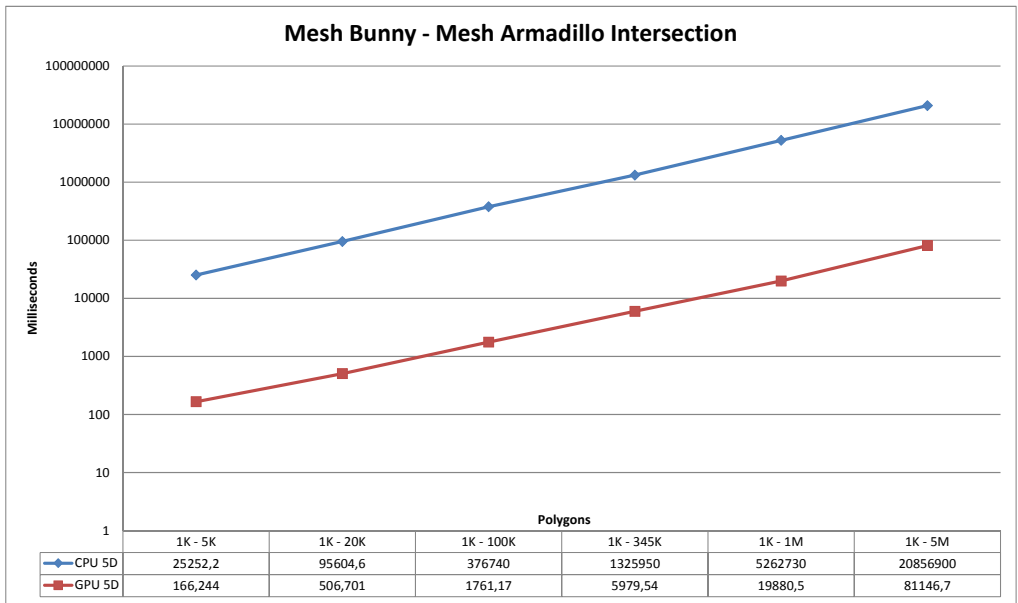
Bunny - Armadillo mesh sizes	Bunny colliding triangles	Armadillo colliding triangles
1K - 5K	187	491
1K - 20K	185	996
1K - 100K	194	2215
1K - 345K	195	3968
1K - 1M	107	5498
1K - 5M	191	15063

The table corresponds to collisions of the Bunny – Armadillo meshes (Figure 5.6) and measured times (Figure 5.5). Intersected triangles are bright yellow (*Bunny*) and pink (*Armadillo*). The GPU-5D implementation is between one and two orders of magnitude faster.

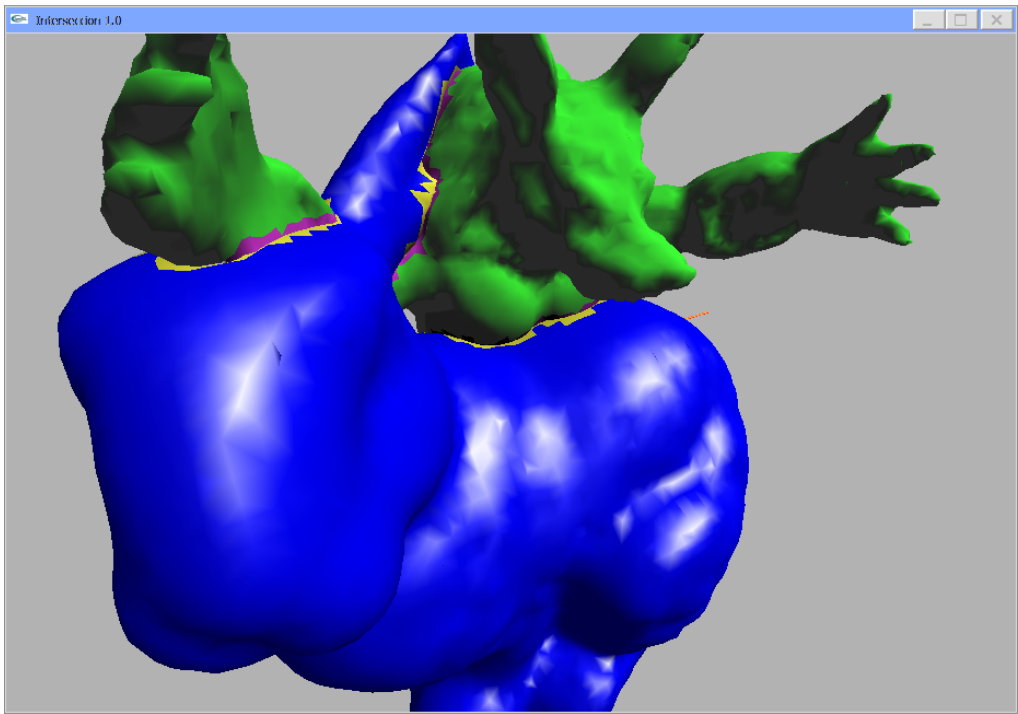
### 5.3.4 Sphere – Sphere collisions

In this setup, 500, 1K, 2.5K, 5K and 10K randomly generated spheres are intersected against each other, as seen in Figure 5.7, with times shown in Figure 5.8. Colliding spheres are shaded in orange. Again, our GPU-5D implementation is between one and two orders of magnitude faster.

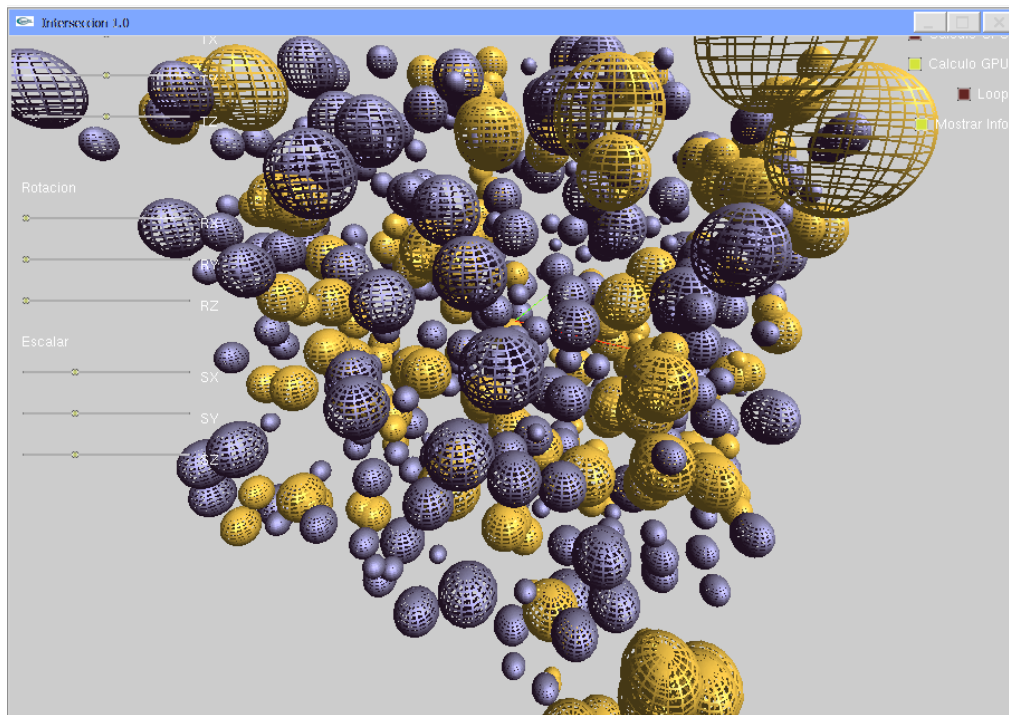




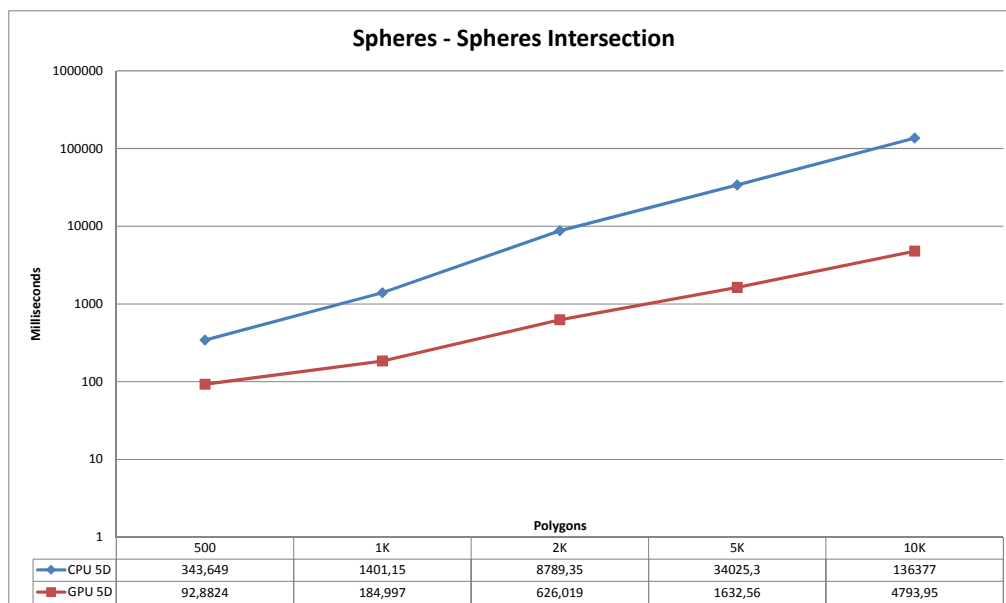
**Figure 5.5** *Bunny mesh - Armadillo Intersection times.*



**Figure 5.6** *Bunny mesh - Armadillo mesh Intersection. Colliding polygons shown in yellow (bunny) and purple (armadillo).*



**Figure 5.7** *Spheres - Spheres Intersection.*



**Figure 5.8** *Spheres - Spheres Intersection times.*

## 5.4 Conclusions from this chapter

We have derived a unified treatment of collisions detection in conformal space from euclidean ( $\mathbb{R}^3$ ) to conformal space ( $\mathbb{R}^{4,1}$ ), sharing a parallel GPU implementation of core CUDA kernels implementing realtime collision detection, defined as algebraic operations that determine intersections among primitives at interactive rates. Throughput is increased by two or more orders of magnitude in collision benchmarks among known mesh models, computed in blind *all vs. all* manner without any bounding volume pre-filtering,

Since our model does not use any pre-filtering techniques, it clearly signals that these values are to be considered as *absolute upper limits on a worst case scenario* and that radical performance improvements of logarithmic nature will be observed when incorporating higher Bounding Volume Hierarchies and other early pruning accelerating techniques to the GPU programming.

The corresponding research and its results have been published in the following references: [\[112\]](#),[\[111\]](#)

# Tangent-space haptic atlases



*A lot of people are afraid of heights.  
Not me, I'm afraid of widths.  
Steven Wright*

**T**His chapter describes the haptic rendering procedure described in Theoktisto *et al* [142, 143], which postulates a new method that dresses triangle meshes with image-based composite rugosity mesostructures "coats" or shells, built out of heightfield displacement textures and normal maps; haptic rendering relies on collisions computed first in conformal  $\mathbb{R}^{4,1}$  space against a low resolution mesh of prisms, and later against the appropriate mesostructure; and visualization is achieved rendering the same mesostructure.

## 6.1 Tangent-space mesostructure atlas generation

Tangent space is defined algebraically for a point  $P$  in an  $n$ -dimensional compact manifold  $M$  by attaching to  $P$  a copy of  $\mathbb{R}^n$  tangential to  $M$  [152], that is, the tangent space of  $M$  at  $P$  is denoted by  $\mathbb{T}_P M$ . For a continuous smooth curve  $\Gamma$  passing through  $P$ , its derivative  $\Gamma'$  is a vector in  $\mathbb{T}_P M$ .

More practically, tangent space [72] is the  $(U, V)$  coordinate system in which texture coordinates are specified for shading. Different faces may have separate tangent space systems. When doing texturing,  $U$  and  $V$  change across a face, and *tangent space* is then the local 2D coordinate system of the face plane. For computing tangent space in 3D surfaces [85], a  $Z$ -axis is needed as well, which takes into the face normal  $\mathbf{n}$ . As usual, for shading purposes, normals are associated to points in the surface itself, so each point  $P_j$  has its own tangent space basis vectors  $\mathbb{T}_{P_j}(\mathbf{t}_j, \mathbf{b}_j, \mathbf{n}_j)$ , where  $\mathbf{t}_j$  and  $\mathbf{b}_j$  are the 2D *tangent* and *bitangent* parametric vectors along the surface and  $\mathbf{n}_j$  is the point's surface normal. Only  $\mathbf{t}_j$  and  $\mathbf{n}_j$  are needed, since the bitangent can be calculated as  $\mathbf{b}_j = \mathbf{n}_j \times \mathbf{t}_j$ .

### 6.1.1 Per-face normal-depth atlases

The local nature of the approach presented in Section 4.2 allows loading arbitrary and unrelated displacement textures and normals on top of a coarse mesh, requiring a special treatment of edge bands when crossing triangles' edges. In the typical case any vertex is ancillary to an average of 6 faces, all of them participating in the blending. This gets very cumbersome if the number of triangles increase.

We now present a global method that uses an atlas of rugosity mesostructures that renders as much as possible surface detail from a dense mesh, capturing geometric information into a multi-

channel texture, which is then used as a “coat” layered onto a much decimated simpler mesh (derived from the former) with all major features preserved. The decimation is obtained by a Quadric Edge Collapse Decimation algorithm [54, 130] using quadric error metrics for optimal polygon reduction and surface fitting, as implemented in the MeshLab software [148].

We represent as much fine geometric detail (considerations regarding surface holes will be described later) using atlases of hybrid rugosity mesostructures (HyRMAs). The process is analogous to using a virtual sharp knife to “slice” triangular prismatoid wedges off the surface of a 3D object until no part of the original surface is left, just a polyhedral skeleton with a lot less detail. Object reconstruction is performed then by “gluing” the triangular wedges back as they were originally.

### 6.1.2 Global-to-local per-face volume warping

A preprocessing step is necessary to obtain the HyRMAs from  $M_f$ , as a high resolution wedge-based multichannel textures in tangent-space computed out of surface differences between the two meshes, encoding local geometry at some precision.

Starting from a dense mesh  $M_f$  showing fine geometric detail, we derive a much simpler mesh  $M_s$  applying a feature-preserving decimation algorithm. Mesh  $M_s$  need not be optimal in size (a separate subject by itself [75]), it only needs to preserve relevant major surface features (ridges and depressions) within some error metric (such as minimizing a distance field). Relief is provided by piecewise prismoid chunks composed of heightfield displacements, normalmaps and other information that are placed on top of each triangle.

Meshes are loaded and superimposed congruently (same size, origin and orientation). A triangular prismoid  $\Pi_k$  is grown enclosing each triangular face  $T_k$  from mesh  $M_s$  alongside its vertices’ normals (both up and down), as shown in Figure 6.1(a). This procedure will triple the number of vertices, and multiply ninefold the number of faces.

Prismoid  $\Pi_k$  maps in normalized barycentric coordinates to a corresponding *uniform* (regular) prism  $\Pi'_k$  (shown in Figure 6.1(b)), grown *orthogonally* up and down from the same triangle  $T_k$ . The mapping establishes *per vertex* invertible warps  $W_i$  from vertices  $H_i$  in volume  $\Pi_k$  to vertices  $H'_i$  in volume  $\Pi'_k$ , so  $W_i(H_i) = H'_i$  and  $W_i^{-1}(H'_i) = H_i$ .

### 6.1.3 Building parametrically equivalent prismoids

A (model dependent) maximum distance  $d$  is calculated by a binary search of a minimum prism height that ensures that all surface geometry of mesh  $M_f$  will be enclosed by prismoids. Top and bottom lids of each will be at  $t^* = 1$  and  $t^* = 0$ .

Since  $P_j|_{t^*=1/2} = P'_j|_{t^*=1/2} = P_j$ , both middle prism triangles (from mesh  $M_s$ ) are equal, that is,  $\Delta P_0(t^*)P_1(t^*)P_2(t^*)|_{t^*=1/2}$  and  $\Delta P'_0(t^*)P'_1(t^*)P'_2(t^*)|_{t^*=1/2}$  are identical. The prisms  $\Pi_k$  and  $\Pi'_k$  they generate are easily constructed as follows:

- To build the regular prism  $\Pi'_k$ , the triangle  $T_k = \triangle P_0 P_1 P_2$  is displaced up and down its face normal  $\mathbf{n}$  by  $\frac{d}{2}$ , with

$$U'_j = P_j - \frac{d}{2} \mathbf{n}, \quad V'_j = P_j + \frac{d}{2} \mathbf{n} \quad (6.1.1)$$

- To build prismoid  $\Pi_k$ , the *plane* of triangle  $T_k = \triangle P_0 P_1 P_2$  is also displaced by  $\frac{d}{2}$  up and down its own face normal  $\mathbf{n}$ , but then new corners  $V_j$  and  $U_j$  are computed from vertices  $P_j$ 's by intersecting the new planes along its vertex normals  $\mathbf{n}_j$ 's, and thus creating all edges of the prismoid:

$$\begin{aligned} V_j - U_j &= r_j \mathbf{n}_j & r_j &= \frac{d}{\mathbf{n} \cdot \mathbf{n}_j} \\ U_j &= P_j - \frac{r_j}{2} \mathbf{n}_j, & V_j &= P_j + \frac{r_j}{2} \mathbf{n}_j \end{aligned} \quad (6.1.2)$$

When projecting the distance vector from this vertex  $H_i = H(t, \alpha, \beta)$  to any of the  $U_j$ 's vertices of mesh  $M_s$  against face normal  $\mathbf{n}$ , the length of this projected vector corresponds to the (relative)  $t^*$  parametric “height” value from its face of mesh  $M_s$ , as expressed in Equation 6.1.3.

$$\begin{aligned} P_j(t^*) &= U_j + t^*(V_j - U_j) & \Rightarrow & P_j(t^*) = P_j + (t^* - \frac{1}{2})(r_j \mathbf{n}_j) \\ P'_j(t^*) &= U'_j + t^*(V'_j - U'_j) & \Rightarrow & P'_j(t^*) = P'_j + (t^* - \frac{1}{2})(d \mathbf{n}) \end{aligned} \quad (6.1.3)$$

$$\text{Let } t \in \mathbb{R}[-\frac{1}{2}, +\frac{1}{2}] = t^* - \frac{1}{2}$$

Then  $t$  may be computed indistinctly at any value  $j$  (0, 1, or 2)

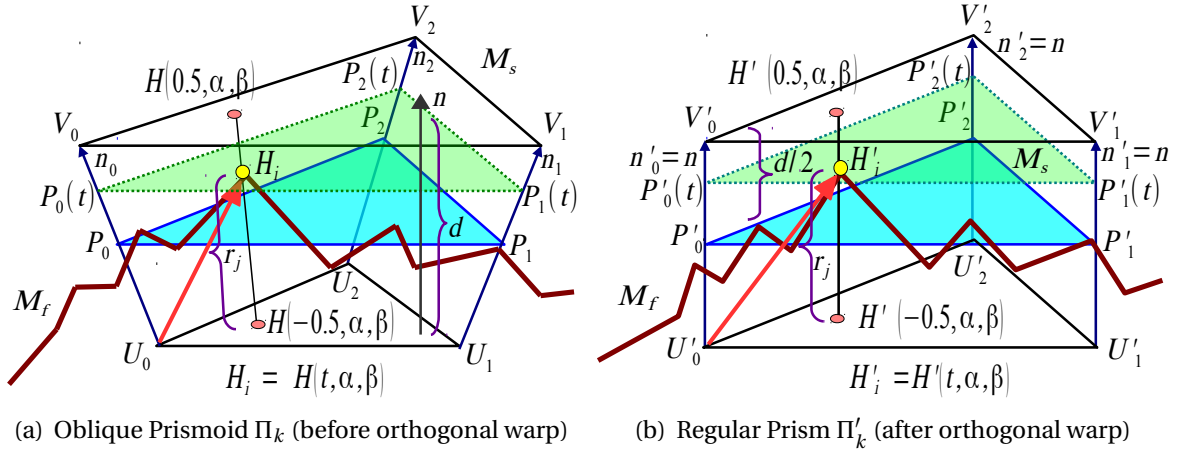
(6.1.4)

$$t = \frac{(H_i - U_j) \cdot \mathbf{n}}{(V_j - U_j) \cdot \mathbf{n}} - \frac{1}{2} \equiv \frac{(H_i - P_j) \cdot \mathbf{n}}{(V_j - U_j) \cdot \mathbf{n}} \Big|_{j=0} \Rightarrow t = \frac{1}{d} (H_i - P_0) \cdot \mathbf{n}$$

As seen on Equation 6.1.4, for vertices  $H_i$  and  $H'_i$  within the prismoids, their common parametric height  $t$  gets mapped into  $\mathbb{R}[-\frac{1}{2}, +\frac{1}{2}]$  interval, and its sign indicates whether the vertex is *over* ( $t > 0$ ), *on* ( $t = 0$ ), or *under* ( $t < 0$ ) base triangle  $T_k$ . There is no need to actually *build* the prismoids in Equations 6.1.1 and 6.1.2, except for visual reference, since  $t$  is computed easily based only on  $H_i$ ,  $P_0$ ,  $\mathbf{n}$ , and the maximum prism height  $d$ .

#### 6.1.4 Obtaining barycentric coordinates applying the “*perp*” operator

For a given  $t$ , and known vertices  $\{P_0, P_1, P_2\}$ , we have all the information needed to compute the  $\langle \alpha, \beta, \gamma \rangle$  triads. Volumes  $\Pi_k$  and  $\Pi'_k$  are parametrically equivalent (Equation 6.1.5), so we solve a  $2 \times 2$  linear system to obtain the  $\langle \alpha, \beta \rangle$  pair, and then  $\gamma$  by subtraction from  $1 - \alpha - \beta$ .



**Figure 6.1** Heightfield Displacement Computation: Obtain transformations  $W_i$ , so each vertex  $H_i$  in the  $M_F$  mesh that is either within the face's prism or immediately adjacent to such a vertex, is transformed into the orthogonal prism.

Let  $\mathbb{P}$  be a 3D plane formed by the three nonaligned vertices of triangle  $\Delta P_0(t)P_1(t)P_2(t)$  (and  $\mathbb{P}'$  for triangle  $\Delta P'_0(t)P'_1(t)P'_2(t)$ ). Expressing points  $H_i \in \mathbb{P}$  and  $H'_i \in \mathbb{P}'$  within each prismoid in areal barycentric coordinates yields

$$\begin{aligned}
 H_i &= H(t, \alpha, \beta, \gamma) = \alpha P_1(t) + \beta P_2(t) + \gamma P_0(t) \\
 H'_i &= H'(t, \alpha, \beta, \gamma) = \alpha P'_1(t) + \beta P'_2(t) + \gamma P'_0(t) \\
 H(t, \alpha, \beta) - P_0(t) &= \alpha [P_1(t) - P_0(t)] + \beta [P_2(t) - P_0(t)] \\
 H'(t, \alpha, \beta) - P'_0(t) &= \alpha [P'_1(t) - P'_0(t)] + \beta [P'_2(t) - P'_0(t)] \\
 \gamma &= 1 - \alpha - \beta, \quad \alpha, \beta, \gamma \in \mathbb{R}[0..1]
 \end{aligned} \tag{6.1.5}$$

Let

$$\mathbf{x} = P_1(t) - P_0(t), \quad \mathbf{y} = P_2(t) - P_0(t), \quad \mathbf{n} = \mathbf{x} \times \mathbf{y}.$$

then

$$\begin{aligned}
 \mathbf{z} &= H(t, \alpha, \beta) - P_0(t), \quad \text{so } \mathbf{z} = \alpha \mathbf{x} + \beta \mathbf{y}, \quad \text{with} \\
 \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{P} &\implies \mathbf{n} \cdot \mathbf{x} = \mathbf{n} \cdot \mathbf{y} = \mathbf{n} \cdot \mathbf{z} = 0
 \end{aligned}$$

To speed up computation of the barycentric coordinates, we apply the linear “*perp*” operator ( $\perp$ ) described by Hill [51], a 90° CCW *rotor* for 2D planes. We extend it to the 3D plane  $\mathbb{P}$  having unitary normal  $\mathbf{n}$  from triangle  $\Delta P_0(t)P_1(t)P_2(t)$ .

For any vector  $\mathbf{s} \in \mathbb{P}$ , the cross product  $\mathbf{n} \times \mathbf{s} = \mathbf{s}^\perp$  yields a vector perpendicular to  $\mathbf{s}$  in the same

plane,  $\mathbf{s}^\perp \in \mathbb{P}$ , with the following properties

$$\begin{aligned} (\mathbf{as})^\perp &= \mathbf{as}^\perp, & \mathbf{s}^{\perp\perp} &= -\mathbf{s}, & |\mathbf{s}| &= |\mathbf{s}^\perp|, & \mathbf{s} \cdot \mathbf{s}^\perp &= 0, & \mathbf{s} \times \mathbf{s}^\perp &= \mathbf{n}, \\ \mathbf{n} \cdot \mathbf{s} &= 0, & \mathbf{n} \cdot \mathbf{s}^\perp &= 0, & \mathbf{s}^\perp \cdot \mathbf{t} &= -\mathbf{s} \cdot \mathbf{t}^\perp, & (\mathbf{as} + \mathbf{bt})^\perp &= \mathbf{as}^\perp + \mathbf{bt}^\perp \end{aligned}$$

for vectors  $\mathbf{n}, \mathbf{s}, \mathbf{t}$  and scalars  $a, b$

To obtain the  $\langle \alpha, \beta \rangle$  pair from  $\mathbf{z} = \alpha \mathbf{x} + \beta \mathbf{y}$ , we apply separate dot products (by  $\mathbf{x}^\perp$  and  $\mathbf{y}^\perp$ ) at both sides:

$$\begin{aligned} \mathbf{z} \cdot \mathbf{y}^\perp &= (\alpha \mathbf{x} + \beta \mathbf{y}) \cdot \mathbf{y}^\perp = \alpha \mathbf{x} \cdot \mathbf{y}^\perp + \beta \mathbf{y} \cdot \mathbf{y}^\perp \Rightarrow \mathbf{z} \cdot \mathbf{y}^\perp = \alpha \mathbf{x} \cdot \mathbf{y}^\perp \\ \mathbf{z} \cdot \mathbf{x}^\perp &= (\alpha \mathbf{x} + \beta \mathbf{y}) \cdot \mathbf{x}^\perp = \alpha \mathbf{x} \cdot \mathbf{x}^\perp + \beta \mathbf{y} \cdot \mathbf{x}^\perp \Rightarrow \mathbf{z} \cdot \mathbf{x}^\perp = \beta \mathbf{y} \cdot \mathbf{x}^\perp \end{aligned}$$

Replacing back  $[\mathbf{x}^\perp, \mathbf{y}^\perp]$  by  $[\mathbf{n} \times \mathbf{x}, \mathbf{n} \times \mathbf{y}]$ , and then again  $\mathbf{n}$  by  $\mathbf{x} \times \mathbf{y}$ , the introduced vectors  $\mathbf{x}^\perp, \mathbf{y}^\perp$  and  $\mathbf{n}$  are neatly removed:

$$\begin{aligned} \alpha &= \frac{\mathbf{z} \cdot \mathbf{y}^\perp}{\mathbf{x} \cdot \mathbf{y}^\perp} = \frac{\mathbf{z} \cdot (\mathbf{n} \times \mathbf{y})}{\mathbf{x} \cdot (\mathbf{n} \times \mathbf{y})} = \frac{\mathbf{z} \cdot ((\mathbf{x} \times \mathbf{y}) \times \mathbf{y})}{\mathbf{x} \cdot ((\mathbf{x} \times \mathbf{y}) \times \mathbf{y})} \\ \beta &= \frac{\mathbf{z} \cdot \mathbf{x}^\perp}{\mathbf{y} \cdot \mathbf{x}^\perp} = \frac{\mathbf{z} \cdot (\mathbf{n} \times \mathbf{x})}{\mathbf{y} \cdot (\mathbf{n} \times \mathbf{x})} = \frac{\mathbf{z} \cdot ((\mathbf{x} \times \mathbf{y}) \times \mathbf{x})}{\mathbf{y} \cdot ((\mathbf{x} \times \mathbf{y}) \times \mathbf{x})} \end{aligned}$$

Finally, applying the known expression for calculating the triple cross product expansion,  $(\mathbf{i} \times \mathbf{j}) \times \mathbf{k} = (\mathbf{i} \cdot \mathbf{k})\mathbf{j} - (\mathbf{j} \cdot \mathbf{k})\mathbf{i}$ , and after more substitutions and rearrangements, we obtain elegant complementary expressions for  $\alpha$  and  $\beta$  using a minimum of dot products:

$$\begin{aligned} \alpha &= \frac{(\mathbf{x} \cdot \mathbf{y})(\mathbf{y} \cdot \mathbf{z}) - (\mathbf{y} \cdot \mathbf{y})(\mathbf{x} \cdot \mathbf{z})}{(\mathbf{x} \cdot \mathbf{y})(\mathbf{x} \cdot \mathbf{y}) - (\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{y})} \Rightarrow \alpha = (\mathbf{Uy} - \mathbf{Ax}) \cdot \mathbf{z} \\ \beta &= \frac{(\mathbf{x} \cdot \mathbf{y})(\mathbf{x} \cdot \mathbf{z}) - (\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{z})}{(\mathbf{x} \cdot \mathbf{y})(\mathbf{x} \cdot \mathbf{y}) - (\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{y})} \Rightarrow \beta = (\mathbf{Ux} - \mathbf{By}) \cdot \mathbf{z} \end{aligned}$$

$$U = \frac{\mathbf{x} \cdot \mathbf{y}}{D} \quad A = \frac{\mathbf{y} \cdot \mathbf{y}}{D} \quad B = \frac{\mathbf{x} \cdot \mathbf{x}}{D} \quad (6.1.6)$$

$$D = (\mathbf{x} \cdot \mathbf{y})(\mathbf{x} \cdot \mathbf{y}) - (\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{y})$$

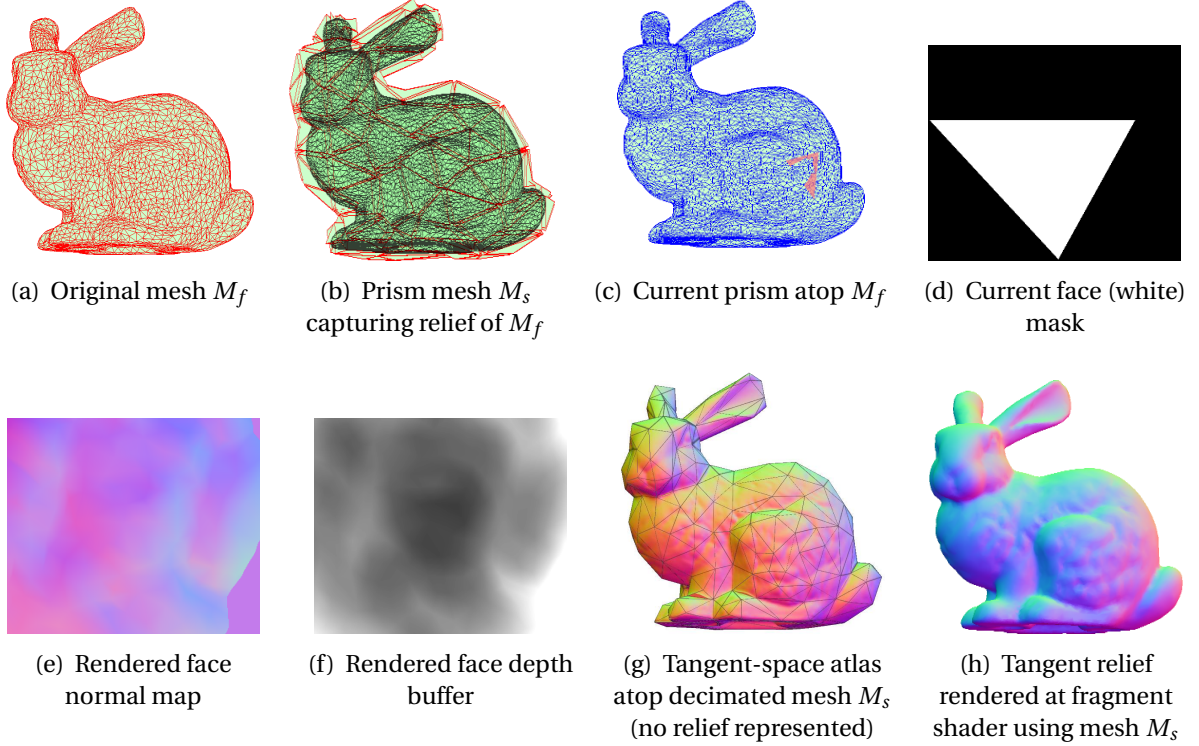
$$H'_i = H'(t, \alpha, \beta) = P_0 + \alpha \mathbf{x} + \beta \mathbf{y} + t d \mathbf{n}$$

Equations 6.1.4 and 6.1.6 allow fast parallel computation of local coordinates  $\langle t, \alpha, \beta \rangle$  using only dot products. Values  $D, U, A$ , and  $B$ ,  $(\mathbf{Uy} - \mathbf{Ax})$  and  $(\mathbf{Ux} - \mathbf{By})$  are calculated for each  $\mathbf{z}$  vector, leaving five simple dot products to compute for each  $H_i$  vertex within the prismoid. From there a quick interpolation yields  $H'_i = H'(t, \alpha, \beta)$  at maximum prism height  $d$ .

By construction, it follows that  $G^0, G^1$  and  $G^2$  continuity are guaranteed across prismoids, since



a point on the shared edge of two adjacent faces of  $M_s$  will be mapped to the same height at both sides, and the computed normals for edge points at both sides will be the same. These properties ensure that when rendering the simple mesh  $M_s$  with the superimposed haptic atlas, the transitions across prismoids will be perceived as an artifact-free continuous surface.



**Figure 6.2** Complete sequence representation of tangent mesostructure atlas construction and image-based relief generation for visualization and real-time haptic rendering.

## 6.2 Tangent-space texture atlas generation using GPU

To obtain the barycentric coordinates for texture sampling, the prism is orthogonalized, warping all vertices  $H_i$  enclosed by the prism volume  $\Pi_k$  into its corresponding vertex images  $H'_i$  in the regular triangular prism  $\Pi'_k$ , as denoted by Equations 6.1.4, 6.1.5 and 6.1.6, in the procedure seen on Figure 6.2. As long as all face triangles are non-degenerate (a feature of good decimation algorithms), these systems are well-conditioned, which makes feasible obtaining  $W_i^{-1}$ , the inverse warping transformation from tangent to object space.

### 6.2.1 Atlas generation procedure

For every face  $f_k$  (Prismoid  $\Pi_k$ ) of  $M_s$

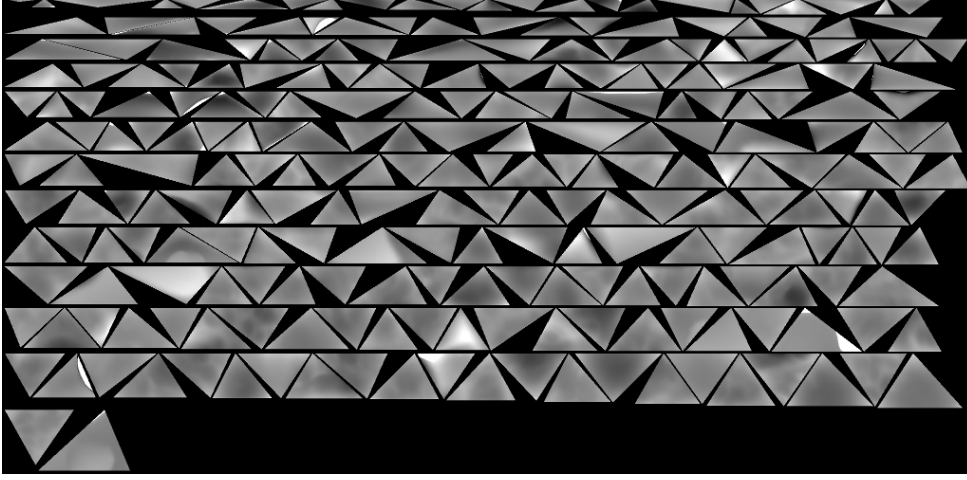
1. A camera gets placed at a pre-computed constant distance  $\frac{d}{2}$  from the center of face  $f_k$  (Figure 6.2(b)), just outside of the top lid of the prism, setup which is replicated at mesh  $M_f$  (Figure 6.2(a)). An orthonormal projection is used to ensure a homogeneous sampling of the relief volume without losing bit precision farther from the camera.
2. Mesh  $M_s$  is then disabled, and we prepare to render mesh  $M_f$  from the same viewpoint, with an overlap border of two pixels, to avoid rendering artifacts at the edge borders (Figure 6.2(c)). All vertices  $H_i$  belonging to  $M_f$  that fall within the corresponding volume of prismoid  $\Pi_k$  are tagged and warped onto its corresponding orthogonal points  $H'_i$  by means of the invertible mapping  $W_i$ , as previously explained in sections 6.1.3 and 6.1.4. The  $\Pi_k$  prismoids are grown slightly to accommodate some neighboring outer vertices and faces from mesh  $M_f$ , to avoid border cases of triangles of the low and high resolution mesh coinciding at edges or vertices).
3. Using the current face as a mask (Figure 6.2(d)), mesh  $M_f$  is simultaneously rendered to multiple texture channels to an off-screen framebuffer using a vertex+fragment GPU shader: the depth buffer in the range between the two lids of the prism, representing the corresponding normal map (Figure 6.2(e)), vertical displacement (Figure 6.2(f)), and the  $uv$  local parametrization coordinates.

An example of how the textured triangles are stored is shown in Figure 6.3, showing the normal vector map in the RGB channel (Figure 6.3(a)) and the relief depth map in the associated  $\alpha$  channel (Figure 6.3(b)). To save memory space and reduce waste, the textured triangles are sorted by largest edge size and stored alternately facing up and down. The corresponding texture  $uv$  coordinates are stored separately. Quality of volume detail will depend on texture size for each rendered triangle in the tangent texture. Texture sizes for individual faces were 256x256 or 512x512, which in reality represent 256x224 and 512x448, the maximum size for the tallest possible equilateral triangle. Except in the sphere models, most triangles are very dissimilar, and are represented by small chunks of pixels in the map. Texture files width range from [256..8192] pixels, but may be limitless in length.

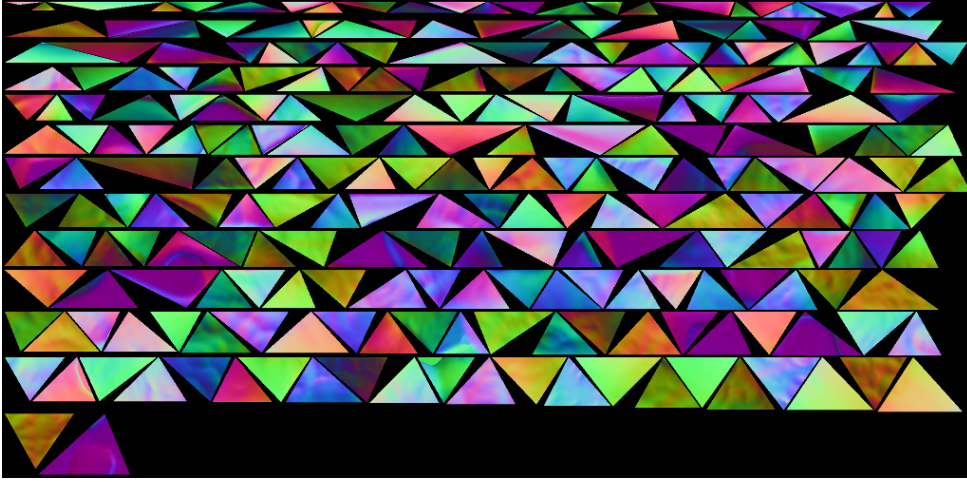
The preprocessing part at the first phase of texture generation takes several seconds depending on mesh density, so it is less than optimal for realtime on-the-fly generation yet. However, after generating the tangent atlas (which only needs to be done once), relief rendering is quite fast.

## 6.3 Haptic rendering using the tangent-space atlas

The procedure for rendering higher visual detail is straightforward, and shown on **Algorithm 6.1**. After loading the lower resolution mesh  $M_s$ , and its tangent-space texture atlas, all  $\Pi_k$  prisms are built and stored for fast retrieval and testing.



(a) Depth atlas for Stanford Bunny (256 faces)



(b) Normal vector atlas for Stanford Bunny (256 faces)

**Figure 6.3** *Tangent texture atlas for the Stanford Bunny, showing ordering by edge length and alternating up and down orientation.*

The haptic probe's position vector,  $G$  (which its coupled orientation vector), is continuously sampled and tested to determine whether it falls inside one of the prisms  $\Pi_k$ , using the collision detection implementation described in Sections 5.2.1 and 5.3.1. When that is the case, the relative height  $t$  and barycentric coordinates  $\alpha, \beta$  are computed,  $G$  is warped and projected orthogonally at the corresponding place in the tangent atlas as

$$G'(t, \alpha, \beta) = \alpha P'_1(t) + \beta P'_2(t) + \gamma P'_0(t) = \alpha P_1 + \beta P_2 + (1 - \alpha - \beta)P_0 + t d \mathbf{n}. \quad (6.3.1)$$

The tangent space atlas height and normal are sampled, obtaining  $H'(t, \alpha, \beta)$  and  $\mathbf{n}(\alpha, \beta)$ . Then, a penetration depth  $D = H'(t, \alpha, \beta) - G'(t, \alpha, \beta)$  is calculated. If positive, it means that the probe is

below the surface, so a real collision has taken place. The penetration depth  $D$  and normal  $\mathbf{n}(\alpha, \beta)$  at that point are returned, as input for a repulsing force to constrain the haptic probe to the surface.

---

**Algorithm 6.1** Haptic rendering with HyRMA Atlas

---

```

1: Input: haptic point  $G$ , mesh  $M_s$ , texture atlas  $HyRMA$ 
2: Output: penetration depth  $D$ , surface normal  $\mathbf{n}$ 
3:
4: loop
5:   sample haptic probe position  $G = (g_x, g_y, g_z)$ 
6:   if  $G$  within some prismoid  $\Pi_k$  from  $M_s$  then                                ▷ Detect collisions against the prismoid mesh
7:                                                                                   ▷ The haptic probe is inside  $\Pi_k$ 
8:     project  $G$  against point  $P_0$  of edge  $\overrightarrow{U_0 V_0}$  of  $\Pi_k$ 
9:     obtain  $t \leftarrow \frac{1}{d} (G - P_0) \cdot \mathbf{n}$ 
10:    obtain warp barycentric coords  $\langle \alpha, \beta \rangle$  of  $G$  as
11:       $G(t, \alpha, \beta)$  within triangle  $T_j = \Delta P_0(t) P_1(t) P_2(t)$ 
12:    map point  $G(t, \alpha, \beta)$  in  $\Pi_k$  to  $\Pi'_k$                                 ▷ its corresponding parametric point
13:     $G'(t, \alpha, \beta) \leftarrow \alpha P_1 + \beta P_2 + (1 - \alpha - \beta) P_0 + t d \mathbf{n}$ 
14:    sample HyRMA pair  $\langle H'(t, \alpha, \beta), \mathbf{n}(\alpha, \beta) \rangle$ 
15:    if  $D \leftarrow H'(t, \alpha, \beta) - G'(t, \alpha, \beta) \geq 0$  then
16:      return pair  $\langle D, \mathbf{n}(\alpha, \beta) \rangle$                                 ▷ A real collision against surface geometry !!!
17:    else
18:      return                                                        ▷ No collisions !!!
19:    end if
20:  end if
21: end loop

```

---

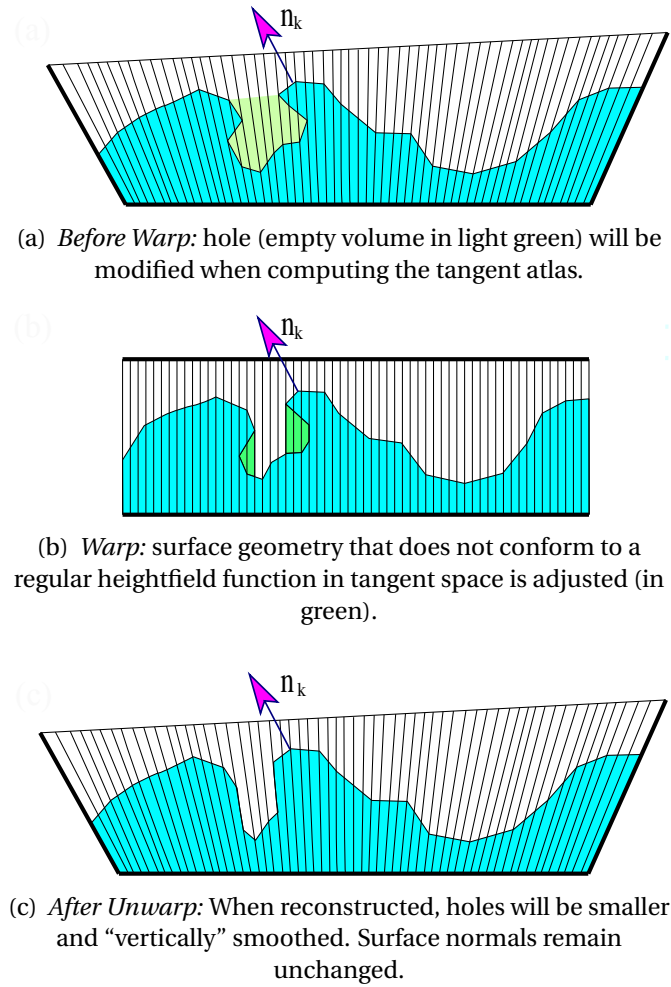
Quality of volume detail will depend on texture size for each rendered triangle in the tangent texture, and a maximum size of 256x256 or 512x512 for each triangle of the coarse mesh is quite adequate for quick and accurate relief rendering.

The creation of a tangent atlas texture is a potentially one-time *lossy* transformation. As can be seen in Figure 6.4, obtaining the atlas will smooth some surface depressions and “concave” subholes (Figure 6.4(a)). These features will be transformed orthogonally and lose subsurface empty volume (Figure 6.4(b)).

When reconstructed, surface holes will be no longer concave (where “concave” is relative to the surface interpolated normals) ((Figure 6.4(c))). This will have minor effects in lighting and shadow generation, depending on the (adjustable) decimation level. Models containing large holes with recesses will be greatly affected, if decimated to very a very coarse mesh.

### 6.3.1 Rendering with a modified Parallax Occlusion Mapping shader

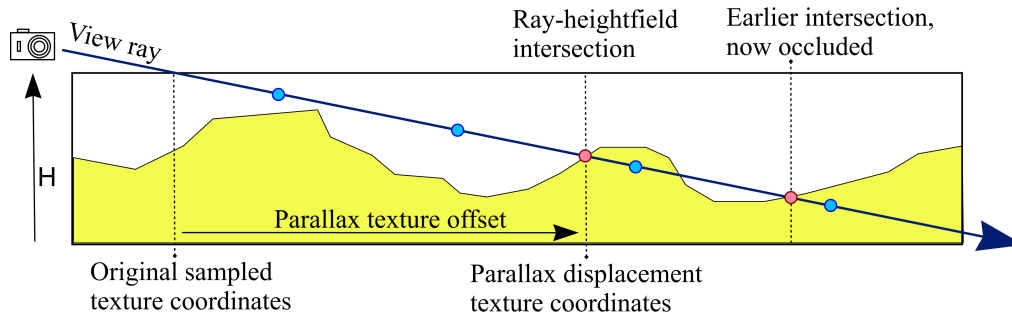
The same HyRMA mesostructure textures may be used to render the scene. A shader was implemented with some modifications over the parallax occlusion shader implemented by Tatarchuk [131], to account for the correct sampling the tangent atlas relief and normal texture. The method is based on a ray-caster acting as a pixel shader, as illustrated in Figure 6.5.



**Figure 6.4** Explanation of potential lossy reconstruction using the tangent atlas warp.

The procedure is as follows:

1. In the vertex shader pass, triangle is identified (all 1+8 triangles belonging to the same prism share one identifier. All necessary variables are setup for the fragment shader pass, as the warped entry and exit points of the view ray intersecting the prism. The outer silhouette of the object is enclosed by the extruded prism.
2. Loop through a linear search, sampling several points back to front from the view ray exit point up to the entry point in the prism. Sampling rate is dynamically adjusted according to the angle between the view ray and the geometric normal of the face.
3. Sample depth and normal from the texture atlas. If ray height is lower than the relief height at the sampling point, search for boundary in the opposite direction (towards the camera), and find the point in which ray hits relief surface.



**Figure 6.5** *Parallax Occlusion Mapping shader fundamentals: the View Ray is transformed into the space of the orthogonal prism, and then sampled repeatedly across the (warped) tangent texture space [blue circles]. Two Ray-heightfield intersections [red circles] are shown; only the frontmost collision is accepted, since it occludes the farther ones.*

4. Obtain parallax displacement and displace pixel the calculated amount, then unwarp the coordinates to obtain the real displacement.
5. Optionally, for colored lights and shadows, add an additional loop to determine which pixels are shaded or in shadow, using each light ray (or shadow ray) instead of the view ray.

In Table 6.1 we show several test models at different resolutions (sphere, bunny, armadillo, Ripoll), the generated texture, their visual framerates and the number of parallax samples for each viewing ray. It is clear that the visual rendering framerate slows as the number of triangles grow. This means also lowering the number of parallax iterations also, since small textures will have less variation of detail and will need less samples per loop. Haptic performance and general interaction may suffer if visual rendering gets below the 60 FPS mark.

## 6.4 Validation and testing procedure

The method has been validated threefold: *geometrically*, by sampling and verifying a trajectory, showing no surface differences between pure geometric and HyRMA-rebuilt models; *statistically*, by obtaining haptic atlas rendering metrics showing optimal haptic/visual rendering framerates; and *perceptually*, by user testing and usability controlled trials measuring accurate haptic sensation of large meshes' fine features at interactive rendering rates.

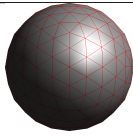
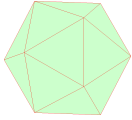

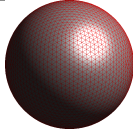
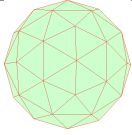


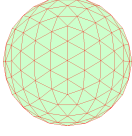













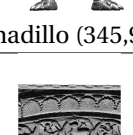
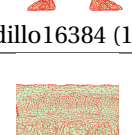
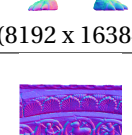
### 6.4.1 Setup

We prepared three separate procedures to implement the aforementioned validations:

- **Test I** detects whether a statistical difference can be established between a trajectory on the dense model and the same trajectory reconstructed out of simulating collisions against the corresponding prisms and tangent-space textures of the simple model.



**Table 6.1** *Test HiRes and LoRes meshes with visual framerates*

HiRes mesh (# of triangles)	LowRes mesh (# of triangles)	Rendered Image (RGBA texture size)	Frames per Second (Max parallax iterations)
 sphere (320)	 icosahedron (20)	 (1024 x 932)	1340 (20)
 sphere4 (5,120)	 sphere1 (80)	 (1024 x 1536 )	645 (10)
 sphere8 (1,310,720)	 sphere2 (320)	 (2048 x 932)	223 (5)
 bunny (144,046)	 bunny512 (512)	 (4096 x 1407)	226 (10)
 bunny (144,046)	 bunny8192 (8,192)	 (4096 x 2048)	171 (5)
 armadillo (32,768)	 armadillo1024 (8,192)	 (4096 x 8192)	155 (5)
 armadillo (345,944)	 armadillo16384 (16,384)	 ((8192 x 16384)	92 (3)
 Ripoll (1,200,000)	 Ripoll (8,192)	 ((8192 x 8192)	120 (3)

- **Test II** checks whether the proposed method is as fast as possible for interactive haptic rates.
- **Test III** performs usability tests on users, applying two separate questionnaires to detect A) Whether faithful user perception is affected by the proposed method and B) Whether users detect haptic resolution differences when compared to less or more detailed visual representations.

### Stimuli

We chose several sets of meshes with a variable number of faces, named using a letter (B=bunny, F=fandisk, S=Sphere, R=Ripoll's portalada) and a number representing the geometric resolution (number of faces of that particular model). Thus,  $B_{1024}$  is the Stanford Bunny model decimated to 1024 faces. We generated separate atlases of heightfield displacement maps, where each triangle is rendered at individual texture blocks of  $256 \times 256$ ,  $512 \times 512$  or  $1024 \times 1024$  texels corresponding to each geometric resolutions.

### Apparatus

- An Intel(R) Core(TM)2 Duo CPU E6550 (2.33GHz, 64 bit, 2 GB RAM) with a NVIDIA Quadro FX 3800/PCIe/SSE2 (1GB RAM) graphics card, under OpenGL/GLSL Shader v. 3.30.
- One PHANTOM Omni® Haptic device, coupled with the OpenHaptics® Toolkit (v. 3) API interface.

### Participants

A total of 14 participants were recruited in separate sessions. Previous to take the tests, they use the haptic device in a free-ranging short session to familiarize themselves with the device and the task at hand.

### Procedure

Each user perform the tests in sequence, answering to a questionnaire that a moderator is administering. At each test, questions measure quantitative and qualitative values of user perception and interaction, to be processed later.

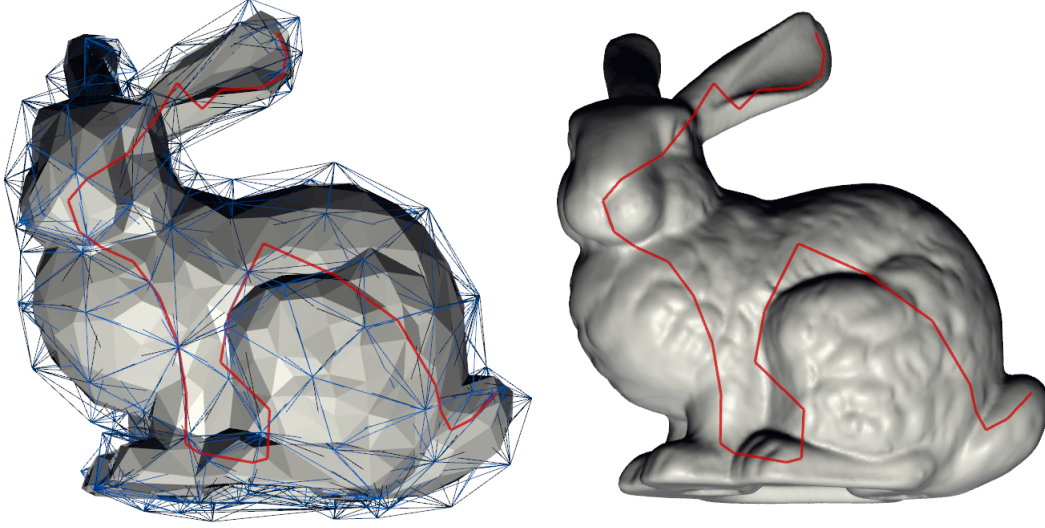
#### 6.4.2 Test I - Trajectory sampling verification

Although the fact that the haptic HyRMA atlas is a invertible mapping from 3D space to tangent space, its wellness may be affected by choosing wrong texture sizes. An easy way to select an appropriate texture size is to sample a trajectory over the surface of of the dense model, and then replay



the same trajectory within the prisms of the simpler model but collisioning against the haptic atlas rendering.

A fitness threshold may be established for measuring how close to the real surface characteristics will be the perceived haptic perception. The trajectories were generated by GPU-based collisions against the meshes geometry as described by Roa *et al* [111]. The original complete Stanford



**Figure 6.6** *Trajectory verification by sampling: A path (in red) intersects mesh  $B_{256}$  (left), then points along this path are sampled in the HyRMA texture to obtain a reconstructed trajectory, which later is compared to corresponding points sampled and warped from the same path intersecting dense mesh  $B_Z$  (right).*

Bunny ( $B_Z$ , 144.046 faces) is paired against three decimated models of 256 faces ( $B_{256}$ ), 512 faces ( $B_{512}$ ), and 1024 faces ( $B_{1024}$ ). A trajectory  $\tau$  is traced along the surface of mesh  $B_Z$ . The same trajectory  $\tau$  is then applied to mesh  $B_{256}$  with all prismoids and HyRMA textures loaded, simulating a haptic interaction (Figure 6.6).

Points along this trajectory are then warped and collisioned in tangent space against the expanded prisms of model  $B_{256}$  (sampling from the atlas texture and clipping against the sampled height). Resulting points are warped back obtaining corresponding reconstructed points.

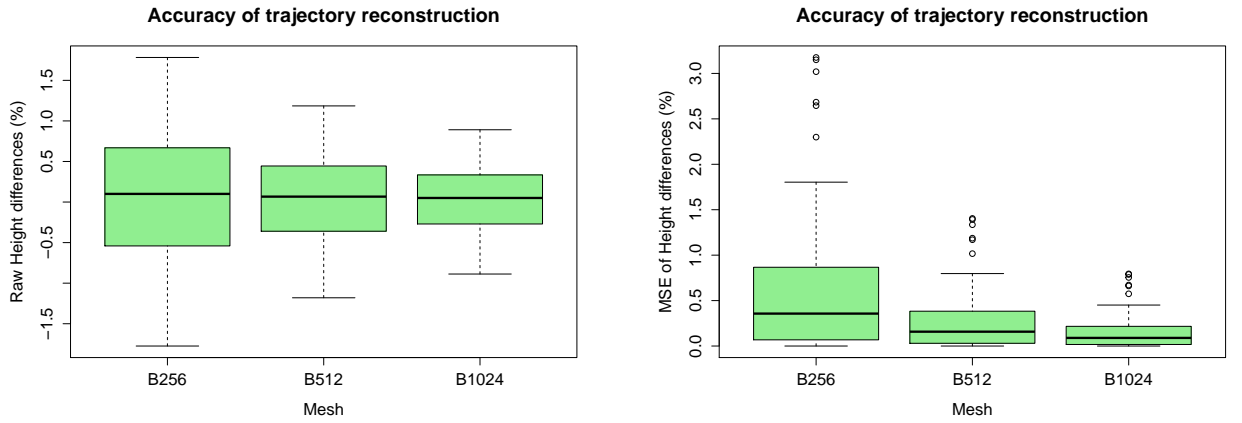
#### 6.4.2.1 Evaluation of results from Test I

Average and Mean Squared Error analysis of height differences of original and reconstructed points is performed to illustrate wellness of the approach. As can be seen from Table 6.2, the mean squared reconstruction error between the two trajectories, which can be extrapolated at the whole approach, is less than 1% of the prism height (Figure 6.7). The boxplots show that the reconstruction error of the simulated models (Mean Square Error analysis of height differences of original and reconstructed points) between the two trajectories is negligible, getting smaller as the LowRes mesh gains

in resolution.

**Table 6.2** *Trajectory verification for the Bunny meshes*

Original LowRes mesh triangles	Original LowRes mesh <i>path segments</i>	Rebuilt HiRes mesh triangles	Rebuilt HiRes mesh <i>path segments</i>	Statistics Mean Sqrd Err	Statistics <i>Std</i> <i>Dev</i>
256	36	256	743	0.61%	0.74%
512	56	512	754	0.27%	0.33%
1024	78	1024	754	0.15%	0.18%



(a) Verification of trajectory accuracy in % height differences. Shaded boxes group the middle 25%-75% of samples.

(b) Accuracy of the Mean Squared Errors from % height differences. The small circles shown above are outliers beyond the  $6\sigma$  interval.

**Figure 6.7** *Accuracy measurement of height reconstruction. Relative height differences get smaller as the dense mesh's resolution increases, and also do outliers.*

### 6.4.3 Test II - Haptic atlas rendering metrics

In order to highlight the advantages of the proposed method for haptic rendering, it is necessary to establish measurement yardsticks for comparison. In this case, we select two different mesh model families built at a range of resolutions (subscript is the number of faces):

**Sphere:** Atlas texture from meshes  $S_{16864}$ , and  $S_{65536}$ , rendered haptically by a  $S_{320}$  mesh.

**Stanford Bunny:** Atlas texture from meshes  $B_{65536}$ , and  $B_{144046}$ , rendered haptically by a  $B_{512}$  mesh.

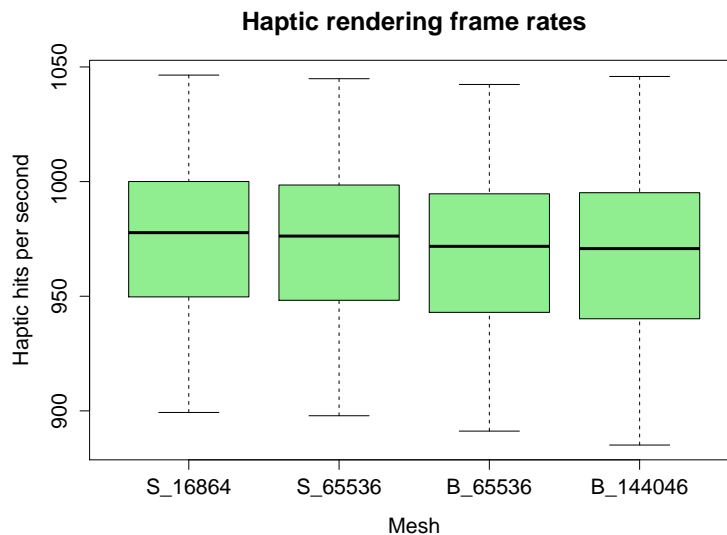
### 6.4.3.1 Evaluation of results from Test II

Spheres were generated recursively from an icosahedron to test the speed in ideal conditions, since complete convex surfaces will not “lose” material, and the triangles are near equilateral and of similar size. For the complete Stanford Bunny, with triangles of all sizes and shapes, it is apparent that the increased number of triangles is offset by the lower individual size of each texture and its parallax rendering.

Haptic surface contacts are rendered as direct collisions between the objects’ geometry and the haptic probe, and haptic framerates (HFPS) (hits/second) are measured (50 continuous measurements). As seen on Table 6.3, average haptic framerates near the 1000 Hz mark for complex models, ensuring continuous, fast and smooth haptic interactions (Figure 6.8). Haptic frame rates decrease slowly with respect to mesh density as expected, but using BVHs and other techniques might maintain optimum rendering performance.

**Table 6.3** *Haptic Framerate measurements (in Hits/Sec)*

Model	$S_{16864}$	$S_{65536}$	$B_{65536}$	$B_{144046}$
HFPS Mean	972.73	971.23	966.62	965.31
(HFPS Std Dev)	(32.32)	(32.29)	(33.21)	(35.31)



**Figure 6.8** *Haptic framerates for meshes growing in complexity. Performance decrease slowly as mesh complexity grows, but still allowing good interactive rates.*

### 6.4.4 Test III - User experience and usability testing

Notwithstanding the correctness of the algebraic derivation and statistical measurements of rendering rates, final user trial protocols are necessary to validate the approach. Two usability questionnaires were applied to 14 participants.

#### 6.4.4.1 Questionnaire A: Protocol for haptic atlas rendering

Questionnaire A applies to 3 known model pairs: *FanDisk* [ $F_{161}$ ,  $F_{12946}$ ], *Stanford Bunny* [ $B_{1024}$ ,  $B_{144046}$ ] and *Ripoll's Portalada* [ $R_{128}$ ,  $R_{1098642}$ ], measuring haptic response and sensation in atlas-based haptic rendering in a scale of [0=imperceptible..4=noticeable]:

Q1. *Do you perceive the principal surface features of the mesh, in correspondence to its visual representation?*

Q2. *Do you perceive haptic "rugosities" that do not have corresponding visual representation?*

Q3. *Do you observe visual relief features that are not perceived haptically?*

**Table 6.4** *Haptic Atlas rendering measurements*

Test A	Q1		Q2		Q3	
Model	Avrg	StdDev	Avrg	StdDev	Avrg	StdDev
$F_{12946}$	3.86	0.36	0.07	0.27	0.00	0.00
$B_{144046}$	3.86	0.36	0.00	0.00	0.07	0.27
$R_{1098642}$	3.64	0.63	0.07	0.27	0.07	0.27

From the results shown Table 6.4 it can be inferred that:

(Q1) Major haptic features are being perceived correctly.

(Q2) No spurious haptic perceptions detected.

(Q3) No visual relief features went unfelt.

This means that for each object model there is an optimal combination of hi and lo resolution meshes that allows capturing and rendering all geometric detail with the best performance.

#### 6.4.4.2 Questionnaire B: User perception testing

Questionnaire B protocol measures haptic resolution disparity decoupled from its visual counterpart, by switching higher or lower haptic atlas resolutions than dictated by the object's geometry.

We use three model pairs: *Sphere-31*, a visual representation of a spherical 1280-face mesh haptically rendered by a spherical 80-face mesh texture; *Sphere-22*, a visual representation of a spherical 320-face mesh haptically rendered by its equivalent spherical 320-face mesh texture; and *Sphere-13*, a visual representation of a spherical 80-face mesh haptically rendered by spherical 1280-face mesh texture. The measurement scale is  $[-2 = \text{visual more decimated than haptic}; 0 = \text{correct visual-haptic correspondence}; 2 = \text{haptic more decimated than visual}]$ .

Q4. *How detailed do you perceive the visual/haptic correspondence of the meshes' surface features?*

**Table 6.5** *User perception of haptic disparity*

Test B	Q4	
Model	Average response	Standard Deviation
<i>Sphere-31</i>	-2.00	0.00
<i>Sphere-22</i>	0.00	0.00
<i>Sphere-13</i>	1.86	0.53

#### 6.4.4.3 Evaluation of results from Test III

As can be seen from Table 6.5, testers duly noted when visual resolution was higher than haptic resolution, when it was the same, and when haptic resolution was greater than visual resolution.

## 6.5 Conclusions from this chapter

We have developed an approach for haptic rendering using an image-based approach, based on a low resolution mesh coupled a highly detailed warped tangent texture atlas.

A preprocessing phase computes the mesostructure atlas from a dense geometric model, by “cutting” local triangular surface chunks within prisms built atop a much lower resolution of the mesh, preserving depth and normals, warping them into tangent space and storing them in an atlas. The procedure for flattening the surface into tangent space warps oblique prisms into regular ones using a clever twist for faster barycentric coordinates computation.

The mesostructure atlas is then used for haptic rendering, by transforming the haptic probe's coordinates into local warped tangent space, and colliding against the texture, effecting an appropriate force at the device that renders good haptic detail at highly interactive rates.

It allows precise surface representation of very fine haptic detail from huge geometric models, reducing processing time, since device collisions against geometry with a high polygon count is necessary for good detail perception. The same atlas is used for visually render the object using a modified Parallax Occlusion Mapping shader, showing correct correspondence between surface

detail rendering and fine features' perception, with a qualification of some loss of mesostructures detail.

The method is validated as follows: *numerically*, by showing no surface differences between pure geometric and HyRMA-rebuilt models; *statistically*, by obtaining optimal haptic/visual rendering framerates; and *perceptually*, in user testing controlled trials measuring accurate haptic sensation of large meshes' fine features at interactive rendering rates.

A recap of the research in this chapter can be found in [143], which is a summary of an extended unpublished version being submitted to a journal for review.

## Experimental notes

All decimated meshes were generated from the original dense meshes using the open source software **MeshLab** [148], developed at the Visual Computing Lab - ISTI - CNR (Italy) for processing, editing and visualizing unstructured 3D triangular meshes.

Statistical analysis were performed using the **R** statistical package [106], with the resulting graphic plots obtained using the **ggplot2** subpackage [154]



## Discussion



*In the beginning was the Word.  
In the end will be the Word.  
Dan Simmons, Hyperion*

When starting any research conducive to the highest academic degree, it is usually unclear to pinpoint in which direction will take the research, or what will be the measurable results before hand. This research is no exception. What began as a search for a faster haptic rendering algorithm for geometric models ended up devising an alternative image-based approach that, while retaining fidelity of surface perception, encoded all of the original geometric detail of choice meshes. Here we present a recapitulation of the conclusions from each chapter, and finalize with some proposals for desirable and promising future work to expand the results presented in this research.

### 7.1 Summary of conclusions

Given that results from this work clearly emerge from three separate knowledge domains, conclusions from the undertaken research has been grouped along similar lines: collaborative capabilities implementation, collision detection, and image-based haptic rendering.

#### 7.1.1 Collaborative Virtual Reality Environments

Based on a characterization of generic collaborative features for VR systems, we have proposed a versatile framework for evolving collaborative capabilities in standalone VR navigators. Our approach incorporates a hybrid distributed user interaction model, multithreaded software components, network communications under a peer-to-peer scalable topology, message passing channels with a custom protocol, and changeable user roles in a multi-camera subscription model.

The framework's development has been validated by a fast porting of the ALICE VR Navigator. The generic cross-platform design allows an easy migration of similar VR applications into complete collaborative virtual reality environments. We extended the collaborative breadth of the framework by including in the Session layer a fourth thread for handling haptic devices, adding high frequency force-feedback events to the interactive session repertoire.



### 7.1.2 Collision Detection's implementation

We have derived a unified treatment of collisions detection in conformal space, based in a reformulation of collision queries from euclidean ( $\mathbb{R}^3$ ) to conformal space ( $\mathbb{R}^{4,1}$ ), sharing a parallel GPU implementation of core CUDA kernels implementing collision detection as algebraic operations that indistinctly determine intersections among lines, circles, polygons and spheres. In the results we increase throughput by two or more orders of magnitude in collision benchmarks among known mesh models, computed in blind *all vs. all* manner without any bounding volume collision pre-filtering. This means that realtime collisions of complex objects in conformal space can be computed at interactive rates.

Since our model does not use any acceleration techniques, it clearly signals that radical performance improvements will be observed when incorporating higher Bounding Volume Hierarchies for early pruning and other accelerating techniques to the GPU programming. Given that any hierarchical approach of bounding volumes will remove large swathes of data from the computations, these values are to be considered as absolute upper limits on a worst case scenario.

### 7.1.3 Tangent Space Texture Atlases for haptic rendering

In a first approach, we have developed a fast and accurate method for rendering local haptic texture in triangle meshes, which allows perception of correct surface details at several resolutions. This extends the use of heightfield haptics beyond the usual field of gigantic terrain textures and allows perceiving higher surface detail without modeling it geometrically. This approach can be used for locally mapping relief textures in triangular meshes and haptically render them in real-time. The method even allows managing LoD in the visual and haptic resolutions for closer approximations, and we have the added benefit of having a repository of assorted HyRMAs, even procedural ones.

The proposed method allows rendering of HyRMA-textured triangle meshes to feel as detailed as very dense geometric meshes, at a fraction of the modeling and processing costs. The sampling's constant (and very low) computation speed accounts for faster collision detection than those reported in the literature, and promises to scale well even for huge models.

An important result is the existence of a definite region for optimal perception of haptic features, with textures having maximum peak heights or valley depths between 5%-15% of an average edge size, which also holds for dynamic characteristics such as groove sense, and left-to-right or right-to-left traversal differences. This includes a blending procedure across edges that smoothes surface traversals even when transitioning between two or more HyRMA textures.

Another result of greater value is the creation of a repository of base meshes and standard HyRMA textures, allowing measuring quantitative and qualitative differences among the haptic rendering algorithms described herein and others that might follow, using the battery of tests devised for this research.

We then extend this result by devising a procedure to scan the fine triangle geometry of dense

meshes, and replacing it with a decimated mesh of larger triangles while capturing most of the perceptible frequency details of the original surface in a blended global HyRMA mesostructure atlas (height displacements, surface normals and other properties such as directed friction and stickiness).

Our global approach for haptic rendering is then image-based, relying on a low resolution mesh and a highly detailed warped tangent texture atlas. The technique is geared for adequate surface representation for interacting with very fine haptic detail from huge geometric models, reducing processing time, since device collisions against geometry with a high polygon count is necessary for good detail perception.

The approach allows reducing this to a calculation involving a low resolution mesh and a texture sampling approach that produces good haptic detail at highly interactive rates, using a pre-processing phase to encode geometry into HyRMA texture. It can be extended further to multiresolution tangent texture atlases, by generating separate mipmap relief textures corresponding to mesh mesostructure at varying resolution. This allows for perceiving more or less haptic detail when zooming in or out the scene.

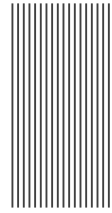
## 7.2 Future work

Following this research we would like to continue working on the field. Further research involves implementing mesh warping operations in a geometry shader to produce the tangent texture atlases “on the fly”, doing away with the map-building pre-processing phase, plus a caching scheme that would avoid (re)computing parts of the mesh most recently travelled.

Another avenue of research to consider is replacing the lossy image-based method for obtaining surface relief from geometric models by a lossless procedure that does not modify concave hole geometry in the surface. A promising approach is obtaining multilevel (3D) relief textures that capture in a flattened 3D atlas all detail enclosed in a predetermined surface thickness, with some compression strategy that defaults to single layer atlas at those surface regions that remain unchanged (within discretization error) when warped.



# Bibliography



*A month in the laboratory can  
often save an hour in the library.*

*E. H. Westheimer*

- [1] C. Andújar, C. Saona-Vázquez, and I. Navazo. LOD visibility culling and occluder synthesis. *Computer Aided Design*, 32(13), 2000. [39](#)
- [2] C. Andújar, C. Saona-Vázquez, I. Navazo, and P. Brunet. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum*, 19(3), August 2000. ISSN 0167-7055. [39](#)
- [3] C. Andújar, P. Brunet, and D. Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, 20(6), 2002. [39](#)
- [4] C. Andújar, M. Fairén, and P. Brunet. Affordable projection system for 3d interaction. In *1st Ibero-American Symposium in Computer Graphics*. University of Minho, Portugal, July 2002. [39](#), [44](#)
- [5] Lionel Baboud and Xavier Décoret. Rendering geometry with relief textures. In *Graphics Interface '06*, Quebec, Canada, 2006. [22](#), [49](#)
- [6] Guibas L. J. Mitchell J. S. Barequet G., Chazelle B. and Tal A. Bintree: A hierarchical representation for surfaces in 3d. In *Computer Graphics Forum*, volume 15, pages 387–396. Blackwell Science Ltd, 1996. [14](#)
- [7] Cagatay Basdogan and Mandayam A. Srinivasan. Haptic rendering in virtual environments. In K. Stanney, editor, *Handbook of Virtual Environments*, chapter 6, pages 117–1343, 2212–2218. London, Lawrence Earlbaum, Inc., 2002. [50](#)
- [8] Eduardo Bayro-Corrochano and Gerik Scheuermann, editors. *Geometric Algebra Computing in Engineering and Computer Science*. Springer, first edition, 2010. ISBN 978-1-84996-107-3. [70](#), [71](#)
- [9] J. F. Blinn. Simulation of wrinkled surfaces. In *Proceedings of SIGGRAPH'78*. ACM, 1978. [49](#)
- [10] C. Bosch, X. Pueyo, S. Mérillau, and D. Ghazanfarpour. A physically-based model for rendering realistic scratches. *Computer Graphics Forum*, 23(3), 2004. [67](#)
- [11] D. Brutzman, M. Zyda, K. Watsen, and Mike Macedonia. Virtual reality transfer protocol (vrtp) design rationale. In *Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality*, Massachusetts Institute of Technology, Cambridge Massachusetts, June 1997. [12](#), [27](#), [29](#)
- [12] Eliya Buyukkaya, Maha Abdallah, and Gwendal Simon. A Survey of Peer-To-Peer Overlay Approaches for Networked Virtual Environments. *Peer-to-peer networking and applications*, 6(3):1 – 25, september 2013. [13](#)

- [13] M. Capps, D. McGregor, D. Brutzman M., and Zyda. Npsnet-v: A new beginning for dynamically extensible virtual environments. *IEEE Computer Graphics and Applications*, 20(5):12–15, 2000. 11
- [14] C. Carlsson and O. Hagsand. Dive: A platform for multi-user virtual environments. *Computers and Graphics*, 17(6):663–669, 1993. 10
- [15] Nathan A. Carr and John C. Hart. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph.*, 21(2):106–131, 2002. ISSN 0730-0301. doi: <http://doi.acm.org/10.1145/508357.508360>. 21
- [16] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313. ACM Press, 2003. ISBN 1-58113-757-5. doi: <http://doi.acm.org/10.1145/945445.945474>. 12
- [17] Seungmoon Choi and Hong Tan. Perceived Instability of VirtualHaptic Texture. I. Experimental Studies. *Presence*, 13(4):395–415, August 2004. 17
- [18] Seungmoon Choi and Hong Z. Tan. Perceived Instability of Virtual Haptic Texture. II. Effect of Collision-Detection Algorithm. *Presence*, 14(4):463–481, August 2005. (not cited)
- [19] Seungmoon Choi and Hong Z. Tan. Perceived Instability of Virtual Haptic Texture: III. Effect of Update Rate. *Presence*, 16(3):263–278, June 2007. 17
- [20] François Conti and Oussama Khatib. Spanning large workspaces using small haptic devices. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2005. 16
- [21] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture*. NVIDIA, USA, 2009. 15
- [22] Michael A. Costa and Mark R. Cutkosky. Roughness Perception of Haptically Displayed Fractal Surfaces. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume 69, pages 1073–1079, 2000. 16
- [23] Sean Curtis, Rasmus Tamstorf, and Dinesh Manocha. Fast collision detection for deformable models using representative-triangles. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games, I3D '08*, pages 61–69, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-983-8. doi: <http://doi.acm.org/10.1145/1342250.1342260>. URL <http://doi.acm.org/10.1145/1342250.1342260>. 14
- [24] Varalakshmi B D, Thriveni J., K R Venugopal, and L M Patnaik. Haptics: State of the Art Survey. *IJCSI International Journal of Computer Science Issues*, 9(3):234–244, 5 2012. 17
- [25] C. Dachsbacher and N. Tatarchuk. Prism Parallax Occlusion Mapping with Accurate Silhouette Generation. In *ACM Symposium on Interactive 3D Graphics and Games (I3D 2007)*, 2007. 22
- [26] Jauvane C. de Oliveira and Nicolas D. Georganas. Velvet: An adaptive hybrid architecture for very large virtual environments. *Presence*, 16(6):555–580, December 2003. 12
- [27] F. Deriggi, M. Kubo, A. Sementille, J. Ferreira, S. dos Santos, and C. Kirner. Corba platform as support for distributed virtual environments. In *Proceedings of IEEE, Virtual Reality'99*, Houston, Texas, March 1999. 11

- [28] Lionel Dominjon, Anatole Lécuyer, Jean-Marie Burkhardt, Guillermo Andrade-Barroso, and Simon Richir. The “Bubble” Technique: Interacting with Large Virtual Environments Using Haptic Devices with Limited Workspace. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2005. 16
- [29] C. Doran and A. Lasenby. *Geometric Algebra for Physicists*. Cambridge, Cambridge, UK, 2009. 70, 71
- [30] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science*. Morgan Kaufmann, San Francisco, CA, 2007. 69, 70, 72
- [31] D. Duce, D. Giorgetti, C. Cooper, J. Gallop, K. Johnson, and C. Seelig. Reference models for distributed cooperative visualization. *Computer Graphics Forum*, 17(4):219–233, 1998. 10, 26
- [32] Ralph Duncan. A Survey of Parallel Computer Architectures. *Computer*, 23:5–16, February 1990. ISSN 0018-9162. doi: 10.1109/2.44904. URL <http://portal.acm.org/citation.cfm?id=78692.78693>. 14
- [33] C. Ericson. *Real Time Collision Detection*. Morgan Kaufmann, San Francisco, CA, 2005. 14
- [34] M. Fairén, P. Brunet, and T. Techmann. MiniVR: A portable virtual reality system. *Computers & Graphics*, 28(2), April 2004. 39, 44
- [35] Fokker Control Systems. *HapticMASTER Installation Manual*. Fokker Control Systems, The Netherlands, December 2002. 16
- [36] Marco Fontana, Emanuele Ruffaldi, Fabio Salsedo, and Massimo Bergamasco. On the Integration of Tactile and Force Feedback. In Dr. Abdulmotaleb El Saddik, editor, *Haptics Rendering and Applications*, chapter 3, pages 47–74. In-Tech OpenAccess, Rijeka, Croatia, January 2012. ISBN 978-953-307-897-7. URL <http://www.intechopen.com/books/haptics-rendering-and-applications/on-the-integration-of-tactile-and-force-feedback->. 16
- [37] M. Franquesa-Niubo and P. Brunet. Collision prediction using mktrees. In *Proceedings of the WSCG 2004, The 12 International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2004*, volume 1, pages 63–70, Pilzen, February 2004. 39
- [38] James Gain and Dominique Bechmann. A survey of spatial deformation from a user-centered perspective. *ACM Trans. Graph.*, 27(4):107:1–107:21, November 2008. ISSN 0730-0301. doi: 10.1145/1409625.1409629. URL <http://doi.acm.org/10.1145/1409625.1409629>. 21
- [39] Joachim Georgii, Jens Krüger, and Rüdiger Westermann. Interactive GPU-based Collision Detection. *IADIS International Journal on Computer Science and Information Systems*, 2(2):162–180, October 2007. ISSN 1646-3692. 14, 77
- [40] Francisco González and Gustavo Patow. Continuity mapping for multi-chart textures. *ACM Trans. Graph.*, 28(5):109:1–109:8, December 2009. ISSN 0730-0301. doi: 10.1145/1618452.1618455. URL <http://doi.acm.org/10.1145/1618452.1618455>. 22
- [41] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtrees: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH ’96, pages 171–180, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: <http://doi.acm.org/10.1145/237170.237244>. URL <http://doi.acm.org/10.1145/237170.237244>. 14

- [42] Stefan Gottschalk. *Collision Queries using Oriented Bounding Boxes*. PhD thesis, University of North Carolina, USA, 2000. 14
- [43] C. Greenhalg, M. Flintham, J. Purbricc, and S. Benford. Application of temporal links: Recording and replaying virtual environments. In *Proceedings of the IEEE VR, 2002*, 2002. 11
- [44] C. Greenhalgh and S. Benford. Massive, a collaborative virtual environment for teleconferencing. *ACM Transactions on Computer, Human Interaction.*, 2(3):239–261, 1994. 10
- [45] A. Gregory, M. Lin, S. Gottschalk, and R. Taylor. H-collide: A framework for fast and accurate collision detection for haptic interaction. In *Proceedings of IEEE Virtual Reality Conference 1999*, pages 38–45. IEEE, 1999. 17
- [46] Markus Gross, Stephan Würmlin, Martin Naef, Edouard Lamboray, Christian Spagno, Andreas Kunz, Esther Koller-Meier, Tomas Svoboda, Luc Van Gool, Silke Lang, Kai Strehlke, Andrew Vande Moere, and Oliver Staadt. Blue-c: a spatially immersive display and 3D video portal for telepresence. *ACM Trans. Graph.*, 22(3):819–827, 2003. ISSN 0730-0301. doi: <http://doi.acm.org/10.1145/882262.882350>. 12
- [47] Louis Guttman. What is Not What in Statistics. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 26(2):81–107, June 1977. ISSN 00390526. doi: 10.2307/2987957. URL <http://www.jstor.org/stable/2987957>. 20
- [48] P. Harting, C. Just, and C. Cruz-Neira. Distributed virtual reality using octopus. In *Proceedings of IEEE Virtual Reality 2001*, pages 53–62, March 2001. 11
- [49] G. Hesina, D. Schmalstieg, A. Fuhrmann, and W. Purgathofer. Distributed open inventor: A practical approach to distributed 3d graphics. In *Proceedings of ACM VRST'99*, pages 74–80, 1999. 11
- [50] D. Hestenes and G. Sobezyk. *Clifford Algebra to Geometric Calculus*. Reidel, Dordreeht, 1984. 69
- [51] F. S. Hill Jr. The pleasures of 'perp dot' products. In Paul Heckbert, editor, *Graphics Gems IV*, pages 138–148. Academic Press, Boston, 1994. 86
- [52] Johannes Hirche, Alexander Ehlert, Stefan Guthe, and Michael Doggett. Hardware accelerated per-pixel displacement mapping. In *Proceedings of Graphics Interface 2004*, GI '04, pages 153–158, 2004. ISBN 1-56881-227-2. URL <http://dl.acm.org/citation.cfm?id=1006058.1006077>. 22
- [53] Chih-Hao Ho, Cagatay Basdogan, and Mandayam. A. Srinivasan. Efficient point-based rendering techniques for haptic display of virtual objects. *Presence*, 8(5):477–491, October 1999. 17
- [54] Hugues Hoppe. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *IEEE Visualization 1999 Conference*, pages 59–66, 1999. 84
- [55] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. Von: a scalable peer-to-peer network for virtual environments. *Network, IEEE*, 20(4):22–31, July 2006. ISSN 0890-8044. doi: 10.1109/MNET.2006.1668400. 12

- 
- [56] R. Hubbard, J. Cook, M. Keates, S. Gibson, T. Howard, A. Murta, A. West, and S. Pettifer. Gnu/-maverik: A micro-kernel for large-scale virtual environments. In *Proceedings of the ACM Symposium on VR Software and Technology*, pages 66–73. ACM Press, 1999. ISBN 1-58113-141-0. doi: <http://doi.acm.org/10.1145/323663.323674>. 12
  - [57] Roger Hubbard and Martin Keates. Real-time simulation of a stretcher evacuation in a large-scale virtual environment. *Computer Graphics Forum*, 19(2), 2000. 16
  - [58] EPIC GAMES INC. Unreal engine development kit, June 2015. URL [www.unrealengine.com](http://www.unrealengine.com). 13, 31
  - [59] P. Jiménez, F. Thomas, and C. Torras. 3D Collision Detection: A Survey. *Computers & Graphics*, 25(2):269 – 285, 2001. ISSN 0097-8493. doi: [http://dx.doi.org/10.1016/S0097-8493\(00\)00130-8](http://dx.doi.org/10.1016/S0097-8493(00)00130-8). 13
  - [60] David E. Johnson and Peter Willemsen. Accelerated Haptic Rendering of Polygonal Models through Local Descent. In *Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)*, 2004. 17
  - [61] J. Kelso, S. G. Satterfield, L. E. Arsenault, P. M. Ketchan, and R. D. Kriz. Diverse: a framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence: Teleoper. Virtual Environ.*, 12(1):19–36, 2003. ISSN 1054-7460. 11
  - [62] Laehyun Kim, Gaurav Sukhatme, and Mathieu Desbrun. A haptic-rendering technique based on hybrid surface representation. *IEEE Computer Graphics and Applications*, pages 66–75, March/April 2004. 17
  - [63] Peter Kipfer. LCP Algorithms for Collision Detection Using CUDA. In Hubert Nguyen, editor, *GPU Gems 3*, pages 723–739. Addison-Wesley, 2007. 14
  - [64] Arthur E. Kirkpatrick and Sarah A. Douglas. Application-based Evaluation of Haptic Interfaces. In *Proceedings of the 10th International Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems (HAPTICS'02)*, 2002. 18
  - [65] Stephan Krause. A Case for Mutual Notification: A Survey of P2P Protocols for Massively Multiplayer Online Games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 28–33, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-132-3. doi: [10.1145/1517494.1517500](http://doi.acm.org/10.1145/1517494.1517500). URL <http://doi.acm.org/10.1145/1517494.1517500>. 12
  - [66] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/359545.359563>. 11
  - [67] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH Construction on GPUs. *Computer Graphics Forum*, 28:375–384, 2009. ISSN 1467-8659. doi: [10.1111/j.1467-8659.2009.01377.x](http://doi.acm.org/10.1111/j.1467-8659.2009.01377.x). 14
  - [68] C. Lauterbach, Q. Mo, and D. Manocha. gProximity: Hierarchical GPU-based operations for collision and distance queries. *Computer Graphics Forum*, 29:419–428, 2010. ISSN 1467-8659. doi: [10.1111/j.1467-8659.2009.01611.x](http://doi.acm.org/10.1111/j.1467-8659.2009.01611.x). 14
  - [69] S. D. Laycock and A. M. Day. A survey of haptic rendering techniques. *COMPUTER GRAPHICS forum*, 26(1):50–65, 2007. 17, 66



- [70] J. Leigh, A. E. Johnson, and T.A. DeFanti. Cavern: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Journal of Virtual Reality Research, Development and Applications*, 2(2):217–237, 1996. 11
- [71] J. Leigh, A. E. Johnson, and T. A. DeFanti. Issues in the design of a flexible distributed architecture for supporting persistence and interoperability in collaborative virtual environments. In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 1–14. ACM Press, 1997. ISBN 0-89791-985-8. doi: <http://doi.acm.org/10.1145/509593.509614>. 26, 29
- [72] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, 3rd Edition*, chapter 7, pages 180–186. Course Technology Press, Boston, MA, United States, 3rd edition, 2011. ISBN 1435458869, 9781435458864. 83
- [73] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 362–371, New York, NY, USA, 2002. ACM. ISBN 1-58113-521-1. doi: <http://doi.acm.org/10.1145/566570.566590>. 21
- [74] David Luebke and Greg Humphreys. How GPU Work. *IEEE Computer Graphics and Applications*, 40:96–100, 2007. ISSN 0018-9162. doi: <http://dx.doi.org/10.1109/MC.2007.59>. 14
- [75] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003. 84
- [76] B. MacIntyre and Feiner S. A distributed 3d graphics library. In *COMPUTER GRAPHICS (Proceedings of ACM SIGGRAPH 98)*, pages 361–370, July 1998. 11
- [77] K. MacLean and M. Enriquez. Perceptual design of haptic icons. In *Proceedings of the Eurohaptics 2003*, Dublin, Ireland, 2003. 16
- [78] Javier Martín and Joan Savall. Mechanisms for Haptic Torque Feedback. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2005. 16
- [79] Michal Masa and Jiri Zara. The general variables concept: A simple step from single to multi-user environment. In *Fifth International Conference on Information Visualisation (IV'01)*, pages 563–567, London, England, UK, July 2001. 10
- [80] Martin Mauve. How to keep a dead man from shooting. In *Lecture Notes in Computer Science*, volume 1905. Springer-Verlag Heidelberg, 2000. 11
- [81] M. Meister and C. A. Wüthrich. On synchronized simulation in a distributed virtual environment. In V. Skala, editor, *WSCG 2001 Conference Proceedings*, 2001. 11
- [82] Francisco J. Melero, Francisco Soler, Pedro Cano, and Juan Carlos Torres. Real-Time Haptic Rendering on Large Models. In O. Rodriguez, F. Serón, R. Joan-Arinyo, J. Madeiras, J. Rodríguez, and E. Coto, editors, *IV Iberoamerican Symposium in Computer Graphics, SIACG 2009*, volume IV, pages 137–144. Sociedad Venezolana de Computación Gráfica, June 2009. ISBN 978-980-12-3682-5. 17
- [83] John L. Miller. Distributed virtual environment scalability and security. Technical Report UCAM-CL-TR-809, University of Cambridge, Computer Laboratory, October 2011. URL <http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-809.pdf>. 13

- 
- [84] Margaret Minsky and Susan Lederman. Simulated haptic textures: Roughness. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume 58, pages 421–426, 1996. 16
- [85] Martin Mittring. Triangle Mesh Tangent Space Calculation. In Wolfgang Engel, editor, *ShaderX4: Advanced Rendering Techniques*, chapter 2, pages 77–89. Charles River Media, Inc., Rockland, MA, USA, 2005. ISBN 1584504250. 83
- [86] Tomas Möller. A fast triangle-triangle intersection test. *J. Graph. Tools*, 2(2):25–30, November 1997. ISSN 1086-7651. doi: 10.1080/10867651.1997.10487472. URL <http://dx.doi.org/10.1080/10867651.1997.10487472>. 14, 76
- [87] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28, October 1997. ISSN 1086-7651. doi: 10.1080/10867651.1997.10487468. URL <http://dx.doi.org/10.1080/10867651.1997.10487468>. 14
- [88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *SIGGRAPH Comput. Graph.*, 22:289–298, June 1988. ISSN 0097-8930. doi: <http://doi.acm.org/10.1145/378456.378528>. URL <http://doi.acm.org/10.1145/378456.378528>. 13
- [89] H. B. Morgenbesser and M. A. Srinivasan. Force shading for haptic shape perception. In *Proceedings of the ASME Intl Mechanical Eng. Congress and Exposition, Dynamic Systems and Control Division*, pages 407–412, 1996. 17
- [90] Hubert Nguyen. *Gpu Gems 3*. Addison-Wesley Professional, first edition, 2007. ISBN 9780321545428. 14
- [91] Matthias Nießner and Charles Loop. Analytic displacement mapping using hardware tessellation. *ACM Trans. Graph.*, 32(3):26:1–26:9, July 2013. ISSN 0730-0301. doi: 10.1145/2487228.2487234. URL <http://doi.acm.org/10.1145/2487228.2487234>. 22
- [92] Scott Nykl, Chad Mourning, and David Chelberg. Interactive Mesostructures. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 37–44, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1956-0. doi: 10.1145/2448196.2448202. URL <http://doi.acm.org/10.1145/2448196.2448202>. 15
- [93] Michael Ortega, Stephane Redon, and Sabine Coquillart. A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Transactions on Visualization and Computer Graphics*, pages 458–469, May/June 2007. 13
- [94] M.A. Otaduy and M.C. Lin. *High Fidelity Haptic Rendering*. Synthesis lectures in computer graphics and animation. Morgan & Claypool Publishers, 2006. ISBN 9781598291148. 20
- [95] Miguel A. Otaduy and Ming C. Lin. Stable and Responsive Six-Degree-of-Freedom Haptic Manipulation Using Implicit Integration. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2005. 16
- [96] Miguel A. Otaduy, Nitin Jain, Avneesh Sud, and Ming C. Lin. Haptic display of interaction between textured models. In *Proceedings of IEEE Visualization 2004*, pages 297–304, Austin, Texas, 2004. 16
- [97] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 27:375–384, 2007. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2007.01012.x. 14

- [98] Simon Pabst, Artur Koch, and Wolfgang Straßer. Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. *Computer Graphics Forum*, 29(5):1605–1612, 2010. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2010.01769.x. URL <http://dx.doi.org/10.1111/j.1467-8659.2010.01769.x>. 15
- [99] Bryson R. Payne, Saeid O. Belkasim, G. Scott Owen, Michael C. Weeks, and Ying Zhu. Accelerated 2d image processing on gpus. In Vaidy S. Sunderam, Geert Dick van Albada, Peter M. A. Sloot, and Jack J. Dongarra, editors, *Computational Science ICCS 2005*, volume 3515 of *Lecture Notes in Computer Science*, pages 256–264. Springer Berlin / Heidelberg, 2005. 14
- [100] Alexander Ploss, Stefan Wichmann, Frank Glinka, and Sergei Gorlatch. From a single- to multi-server online game: A quake 3 case study using rtf. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, ACE '08, pages 83–90, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-393-8. doi: 10.1145/1501750.1501769. URL <http://doi.acm.org/10.1145/1501750.1501769>. 13
- [101] Fabio Policarpo and Manuel M. Oliveira. Relaxed cone stepping for relief mapping. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 18, pages 409–428. Addison-Wesley Professional, 2007. ISBN 978-0321515261. 23
- [102] Fabio Policarpo, Manuel M. Oliveira, and Joao L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 155–162, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-013-2. doi: <http://doi.acm.org/10.1145/1053427.1053453>. 22, 49
- [103] Kristin Potter, David Johnson, and Elaine Cohen. Height Field Haptics. In *Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)*, 2004. 17, 51
- [104] Emil Praun, Wim Sweldens, and Peter Schröder. Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 179–184, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383277. URL <http://doi.acm.org/10.1145/383259.383277>. 21
- [105] Budirijanto Purnomo, Jonathan D. Cohen, and Subodh Kumar. Seamless texture atlases. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 65–74, New York, NY, USA, 2004. ACM. ISBN 3-905673-13-4. doi: <http://doi.acm.org/10.1145/1057432.1057441>. 21
- [106] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL <http://www.R-project.org/>. 101
- [107] D. Rantzau, K. Frank, U. Lang, D. Rainer, and U. Wössner. COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data. In *Proceedings of the 2nd Workshop on Immersive Projection Technology (IPT '98)*, Ames, Iowa, May 1998. 11
- [108] Michel Raynal and Mukesh Singhal. Logical time: Capturing causality in distributed systems. *Computer*, 29(2):49–56, 1996. ISSN 0018-9162. 11
- [109] Christopher Richard and Mark R. Cutkosky. Friction Identification for Haptic Display. In *Proceedings of the 1999 ASME IMECE*, 1999. 16
- [110] Eduardo Roa. Operaciones de cómputo gráfico en el espacio geométrico conforme 5D usando GPU. Master's thesis, Universidad Simón Bolívar, Venezuela, February 2011. <http://www ldc.usb.ve/~vtheok/thesis/gpuconformal.pdf>. 72

- 
- [111] Eduardo Roa and Víctor Theoktisto. Primitives Intersection with Conformal 5D Geometry. In *Proceedings of CIMENICS 2012, IX International Congress on Numerical Methods in Engineering and Applied Sciences, Porlamar, Venezuela.*, pages 389–394, March 2012. ISBN 978-980-7161-03-9. 82, 96
  - [112] Eduardo Roa, Víctor Theoktisto, Marta Fairén, and Isabel Navazo. GPU Collision Detection in Conformal Geometric Space. In *Proceedings of SIACG 2011: Iberoamerican Symposium on Computer Graphics*, pages 153–156, Faro, Portugal, June 2011. EuroGraphics. 82
  - [113] Fabio Rossi and Massimo Bergamasco. “Single Chart Parameterization of Triangle Meshes”. In Doru Talabă and Angelos Amftitis, editors, *Product Engineering: Tools and Methods based on Virtual Reality*, chapter 1, pages 87–97. Springer Science, Heidelberg, 2008. ISBN 978-1-4020-8199-6. 21
  - [114] E. Samur. *Performance Metrics for Haptic Interfaces*. Springer Series on Touch and Haptic Systems. Springer, 2012. ISBN 9781447142256. 20
  - [115] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. Multi-chart Geometry Images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry Processing*, SGP ’03, pages 146–155, Aire-la-Ville, Switzerland, 2003. ISBN 1-58113-687-0. URL <http://dl.acm.org/citation.cfm?id=882370.882390>. 21
  - [116] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, Boston, 2010. 15
  - [117] Paulo Santos, Rodrigo de Toledo, and Marcelo Gattass. Solid height-map sets: Modeling and visualization. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, SPM ’08, pages 359–365, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-106-4. doi: 10.1145/1364901.1364953. URL <http://doi.acm.org/10.1145/1364901.1364953>. 23
  - [118] G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis. Requirements of peer-to-peer-based massively multiplayer online gaming. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 773–782, May 2007. doi: 10.1109/CCGRID.2007.97. 12
  - [119] Christian Schuckmann, Jan Schümmer, and Peter Seitz. Modeling Collaboration using Shared Objects. In *Proceedings of the International ACM SIGGROUP conference on Supporting group work*, Phoenix, Arizona, United States, 1999. 10
  - [120] Jason Shankel. Fast Heightfield Normal Calculation. In *Game Programming Gems 3*, pages 344–348. Charles River Media, Inc., 2002. 60
  - [121] Vladimir Shumskiy. Gpu ray tracing – comparative study on ray-triangle intersection algorithms. In MarinaL. Gavrilova, C.J.Kenneth Tan, and Anton Konushin, editors, *Transactions on Computational Science XIX*, volume 7870 of *Lecture Notes in Computer Science*, pages 78–91. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39758-5. doi: 10.1007/978-3-642-39759-2\_6. URL [http://dx.doi.org/10.1007/978-3-642-39759-2\\_6](http://dx.doi.org/10.1007/978-3-642-39759-2_6). 15, 77
  - [122] Juhani Siira and Dinesh K. Pai. Haptic texturing: A stochastic approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 557–562, April 1996. 16

- [123] K. G. Sreeni, K. Priyadarshini, A. K. Praseedha, and Subhasis Chaudhuri. Haptic rendering of cultural heritage objects at different scales. In *Proceedings of the 2012 international conference on Haptics: perception, devices, mobility, and communication - Volume Part I*, EuroHaptics'12, pages 505–516, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-31400-1. doi: 10.1007/978-3-642-31401-8\_45. URL [http://dx.doi.org/10.1007/978-3-642-31401-8\\_45](http://dx.doi.org/10.1007/978-3-642-31401-8_45). 17
- [124] M. R. Stytz. Distributed virtual environments. *Computer Graphics and Applications*, 16(3): 19–31, 1996. 10
- [125] R. Suselbeck, G. Schiele, and C. Becker. Peer-to-peer support for low-latency massively multiplayer online games in the cloud. In *Network and Systems Support for Games (NetGames), 2009 8th Annual Workshop on*, pages 1–2, Nov 2009. doi: 10.1109/NETGAMES.2009.5446229. 12
- [126] I.E. Sutherland. A head-mounted three-dimensional display. In *AFIPS Conference Proceedings*, volume 33, pages 757–764, 1968. 39
- [127] Mirko Suznjevic, Maja Matijasevic, and Ognjen Dobrijevic. Action specific massive multiplayer online role playing games traffic analysis: Case study of world of warcraft. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '08, pages 106–107, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-132-3. doi: 10.1145/1517494.1517519. URL <http://doi.acm.org/10.1145/1517494.1517519>. 13
- [128] P. Svoboda, W. Karner, and M. Rupp. Traffic analysis and modeling for world of warcraft. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1612–1617, June 2007. doi: 10.1109/ICC.2007.270. 13
- [129] László Szirmay-Kalos and Tamás Umenhoffer. Displacement Mapping on the GPU – State of the Art. *COMPUTER GRAPHICS forum*, 27(6):1567–1592, September 2008. 22
- [130] Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. Practical quad mesh simplification. *Computer Graphics Forum*, 29(2):407–418, 2010. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2009.01610.x. URL <http://dx.doi.org/10.1111/j.1467-8659.2009.01610.x>. 84
- [131] Natalya Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In Marc Olano and Carlo H. Séquin, editors, *SI3D 06*, pages 63–69. ACM, 2006. ISBN 1-59593-295-X. URL <http://dblp.uni-trier.de/db/conf/si3d/si3d2006.html#Tatarchuk06>. 22, 91
- [132] Natalya Tatarchuk. Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 81–112, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: 10.1145/1185657.1185830. URL <http://doi.acm.org/10.1145/1185657.1185830>. 22
- [133] Unity Technologies. Unity3d engine, June 2015. URL [www.unity3d.com](http://www.unity3d.com). 13, 31
- [134] Víctor Theoktisto and Marta Fairén. On Extending Collaboration in Virtual Reality Environments. In Arnaldo de Albuquerque Araújo, João Luiz Dihi Comba, Isabel Navazo, and A. Augusto Souza, editors, *Proceedings of the IEEE SIBGRAPI/SIACG'04, II Iberoamerican Symposium on Computer Graphics 2004*, pages 324–331, Curitiba, Brasil, October 2004. IEEE Computer Society Order Number P2227. ISBN 0-7695-2227-0, ISSN 1530-1834. 47



- [135] Víctor Theoktisto and Marta Fairén. Enhancing Collaboration in Virtual Reality Applications. *Computers & Graphics*, 29(5):704–718, 2005. 47
- [136] Victor Theoktisto, Marta Fairén, and Isabel Navazo. Collaborative Virtual Reality in the ALICE Platform. In *Proceedings of the Spanish Conference in Computer Graphics (CEIG 2003)*, pages 261–276, 2003. 28, 47
- [137] Víctor Theoktisto, Marta Fairén, and Isabel Navazo. ALICE: A Collaborative Virtual Reality Navigator. In César Mendoza and Fabio Ganovelli, editors, *VRIPHYS'04, Workshop on Virtual Reality Interaction and Physical Simulation*, pages 159–169, Colima, Mexico, September 2004. SMCC (Mexican Society of Computer Science). ISBN 970-692-170-2. 41, 47
- [138] Víctor Theoktisto, Marta Fairén, Isabel Navazo, and Eva Monclús. Rendering detailed haptic textures. In *Proceedings of Second Workshop in Virtual Reality Interactions and Physical Simulations (VRIPHYS '05), Pisa, Italy*, 2005. 68
- [139] Víctor Theoktisto, Marta Fairén, and Isabel Navazo. Hybrid rugosity mesostructures (HRMs) for fast and accurate rendering of fine haptic detail. In *Proceedings of XXXV Latin American Computing Conference (CLEI 2009)*, Pelotas, Rio Grande do Sul, Brazil, September 2009. 68
- [140] Víctor Theoktisto, Marta Fairén, and Isabel Navazo. A hybrid rugosity mesostructure (HRM) for rendering fine haptic detail. Technical Report LSI-09-5-R, Universitat Politècnica de Catalunya. Departament de Llenguatges i sistemes informàtics, 2009. 68
- [141] Víctor Theoktisto, Marta Fairén, and Isabel Navazo. A hybrid rugosity mesostructure (HRM) for rendering fine haptic detail. *CLEI Electronic Journal (CLEIej)* ISSN 0717-5000, 13(3):1–12, December 2010. URL <http://www.clei.org/cleiej/paper.php?id=203>. 13
- [142] Víctor Theoktisto, Marta Fairén, and Isabel Navazo. Generalized haptic relief atlas for rendering surface detail. In Sabine Coquillart, Carlos Andújar, Robert S. Laramée, Andreas Kerren, and José Braz, editors, *GRAPP & IVAPP 2013: Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications, Barcelona, Spain, 21-24 February, 2013*, pages 191–196. SciTePress, 2013. ISBN 978-989-8565-46-4. 83
- [143] Victor Theoktisto, Marta Fairén, and Isabel Navazo. Tangent-space mesostructure atlases for accurate real-time haptic rendering. In *XXIV Spanish Conference in Computer Graphics (CEIG'2014)*, pages 1–10, University of Zaragoza, Spain, July 2014. EuroGraphics. 83, 101
- [144] Ville Timonen and Jan Westerholm. Scalable Height Field Self-Shadowing. *COMPUTER GRAPHICS forum, EuroGraphics 2010 issue*, 29(2):723–731, May 2010. 23
- [145] H. Tramberend. Avocado: A distributed virtual reality framework. In *Proceedings of the IEEE Virtual Reality*, 1999. 11
- [146] James M. Van Verth and Lars M. Bishop. *Essential Mathematics for Games and Interactive Applications*. Morgan Kaufmann, San Francisco, 2004. 71
- [147] J. Vince. *Geometric Algebra for Computer Graphics*. Springer, London, UK, 2008. 69
- [148] Visual Computing Lab, ISTI-CNR, Pisa, Italy. MeshLab, 2012. <http://meshlab.sourceforge.net/>. 84, 101

- [149] G. Voss, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scenegraphs and its extension to clusters. In *4th Eurographics Workshop on Parallel Graphics and Visualization, EGPGV02*, Blaubeuren, Germany, 2002. 31
- [150] K. Watsen and M. Zyda. Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments. In *VRAIS '98: Proceedings of the Virtual Reality Annual International Symposium*, page 252. IEEE Computer Society, 1998. ISBN 0-8186-8362-7. 12
- [151] M. Wedlake, K. F. Li, and F. El Guibaly. The NAVL Distributed Virtual Reality System. *Lecture Notes in Computer Science*, 1554:177–193, 1999. ISSN 0302-9743. 11
- [152] Eric W. Weisstein. Tangent space, 2014. URL <http://mathworld.wolfram.com/TangentSpace.html>. 83
- [153] René Weller. *New Geometric Data Structures for Collision Detection and Haptics*. Springer Series on Touch and Haptic Systems. Springer International Publishing, 2013. ISBN 978-3-319-01019-9. doi: 10.1007/978-3-319-01020-5. 17
- [154] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN 978-0-387-98140-6. URL <http://had.co.nz/ggplot2/book>. 101
- [155] S. Wilson, H. Sayers, and M. D. J. McNeill. Using CORBA middleware to support the development of distributed virtual environment applications. In V. Skala, editor, *WSCG 2001 Conference Proceedings*, pages 98–105, 2001. 12
- [156] Dongbo Xiao and Roger Hubbard. Navigation guided by artificial force fields. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1, pages 179–186, April 1998. ISBN 0-201-30987-4. 16
- [157] Amir Yahyavi and Bettina Kemme. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Comput. Surv.*, 46(1):9:1–9:51, July 2013. ISSN 0360-0300. doi: 10.1145/2522968.2522977. URL <http://doi.acm.org/10.1145/2522968.2522977>. 13
- [158] B. Zeleznik, L. Holden, M. Capps, H. Abrams, and T. Miller. Scene-graph-as-bus: Collaboration between heterogeneous stand-alone 3-d graphical applications. In M. Gross and F.R.A. Hopgood (Guest Editors), editors, *EUROGRAPHICS 2000*, volume 19, 2000. 11
- [159] Xinyu Zhang and Young J. Kim. Interactive Collision Detection for Deformable Models Using Streaming AABBs. *IEEE Transactions on Visualization and Computer Graphics*, 13:318–329, March 2007. ISSN 1077-2626. doi: <http://dx.doi.org/10.1109/TVCG.2007.42>. URL <http://dx.doi.org/10.1109/TVCG.2007.42>. 14
- [160] Shanyang Zhao. Toward a taxonomy of copresence. *Presence*, 12(5):445–455, October 2003. 10, 26
- [161] C.B. Zilles and J.K. Salisbury. A constraint-based god-object method for haptic display. In *Proc. IEEE Int'l Conf. Intelligent Robots and Systems*, pages 31–46, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7108-4. 16

*“Beppu (n.) The triumphant slamming shut  
of a book after reading the final page.”*  
Douglas Adams