Real-time Reduced Large-Deformation Models and Distributed Contact for Computer Graphics and Haptics

PhD Thesis

Jernej Barbič

CMU-CS-07-145

August 2007

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Doug L. James, CMU, Chair
Ralph L. Hollis, CMU
Nancy S. Pollard, CMU
Dinesh K. Pai, University of British Columbia

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation (CAREER-0430528), the Boeing Company, the Link Foundation fellowship to Jernej Barbič, and by generous donations from Pixar, Nvidia, Intel, and Maya licenses donated by Autodesk.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: real-time, deformations, model reduction, finite element method, subspace integration, distributed contact, haptics, distance fields, multi-resolution, interruptible, time-critical, simulation, precomputation, computer graphics

I thank my family for love and encouragement,
Daylian Cain, Vahe Poladian and Lucian Lita for friendship,
Carnegie Mellon graphics lab for camaraderie and paper discussions,
committee members for reading and improving the thesis,
and my advisor Prof Doug James for inspiration, support, and the beautiful research problems.

Abstract

This thesis presents novel algorithms for deformable dynamics, collision detection and contact resolution between *reduced* nonlinear 3D deformable objects, for use in interactive computer graphics and haptics. The deformable models investigated are elastic, volumetric, and capable of undergoing large deformations. Each mesh vertex of a general 3D deformable object has three degrees of freedom. Non-interactive computation times result when simulating large-deformation dynamics of such *unreduced* systems (assuming non-trivial geometry). *Reduced* deformable objects are obtained by substituting these general degrees of freedom for a much smaller appropriately defined set of reduced degrees of freedom. This dimensionality reduction can enable much faster simulation times, with some loss of simulation accuracy. Many interesting objects can be well approximated by reduced deformations: swaying bridges, plants, tall buildings, mechanical components (hoses, wires), and human tissue (thigh passively deforming after a jump). The reduced deformable degrees of freedom need to be defined carefully so that they support "typical" large deformations. We present an automatic degree-of-freedom selection algorithm, and an algorithm for fast runtime simulation of the resulting reduced nonlinear dynamics for geometrically nonlinear deformable models.

Real-time deformable objects can be used to provide multi-sensory feedback in emerging real-time applications, such as 6-DOF (force and torque) haptic rendering. It is challenging to perform collision detection and compute contact forces and torques between geometrically complex objects at haptic rates. This thesis presents a CPU-based approach to simulate distributed contact between two (rigid or reduced-deformable) objects with complex geometry. Penalty-based contact forces are resolved using a multi-resolution point-based representation for one object, and a signed-distance field for the other. Our algorithm can adapt the contact force accuracy to both the difficulty of the current contact configuration and the speed of the particular computer. Reduced-deformed distance fields are proposed to support contact between reduced deformable objects. We also expose several important algorithmic details essential for stable and robust 6-DOF haptic rendering. Applications of our work include computer animation and games (including game haptics), CAD/CAM (virtual prototyping), and interactive virtual medicine.

Contents

1 Introduction **21**
- 1.1 Fast deformable models using model reduction . . . 23
- 1.2 Real-time deformable contact and haptic rendering . . . 26
- 1.3 Thesis overview . . . 28
- 1.4 Thesis contributions . . . 29

2 Real-time geometrically nonlinear deformable models using model reduction **33**
- 2.1 Background: Nonlinear deformable object modeling . . . 33
  - 2.1.1 The finite element method . . . 33
  - 2.1.2 Strain and stress . . . 35
  - 2.1.3 Elastic and hyperelastic deformable models . . . 37
  - 2.1.4 Linear materials . . . 38
  - 2.1.5 Nonlinear material models . . . 39
  - 2.1.6 Why the simulation is nonlinear . . . 39
  - 2.1.7 The equations of motion . . . 41
  - 2.1.8 Cubic polynomials for linear materials . . . 42
  - 2.1.9 Offline simulation . . . 45
  - 2.1.10 Static simulations . . . 46
- 2.2 Related work: Real-time deformable objects . . . 47
- 2.3 Related work: Dimensional model reduction in solid mechanics . . . 51
- 2.4 Overview of our approach . . . 52
- 2.5 Mesh generation . . . 53
  - 2.5.1 Choice of elements . . . 53
  - 2.5.2 Free-form deformations . . . 55
- 2.6 The reduced equations of motion . . . 57
  - 2.6.1 Reduced internal forces are cubic polynomials . . . 58
  - 2.6.2 Reduced tangent stiffness matrix entries are quadratic polynomials . . . 59
  - 2.6.3 Precomputing polynomial coefficients . . . 59
  - 2.6.4 Runtime polynomial evaluation . . . 59
- 2.7 Damping . . . 60
- 2.8 Low-dimensional basis selection . . . 61

2.8.1 Mass PCA . . . 62
2.8.2 Sketch-based basis generation . . . 64
2.8.3 Basis from modal derivatives . . . 66
2.8.4 Alternative view on modal derivatives . . . 70
2.8.5 Modal derivatives generalization to second-order ODEs . . . 71
2.8.6 Comparison: modal derivatives vs sketch basis . . . 72
2.8.7 Basis for unconstrained models . . . 73
2.9 The implicit Newmark (subspace) integrator . . . 74
2.9.1 Why not explicit subspace integration? . . . 77
2.9.2 Numerical damping . . . 78
2.9.3 Choice of timestep for implicit Newmark . . . 79
2.10 External forces . . . 80
2.11 Runtime modification of material parameters . . . 80
2.12 Graphical rendering . . . 80
2.12.1 Computing *u*=*Uq* on the GPU . . . 81
2.12.2 Adaptive $u = Uq$ computation for collision detection . . . 82
2.13 Experiments . . . 82
2.13.1 Re-simulation, modal derivatives, sketch . . . 83
2.13.2 Modal derivative basis under progressive r . . . 83
2.14 Real-time interactive demo . . . 88
2.15 Discussion and future work . . . 88

3 Time-critical distributed deformable contact 93
3.1 Background: The Voxmap-PointShell (VPS) method . . . 95
3.1.1 Virtual coupling . . . 98
3.2 Overview of our approach . . . 101
3.3 Related work: Multi-resolution rendering . . . 103
3.4 Related work: Haptic rendering of deformable objects . . . 104
3.5 Penalty-based point-contact model . . . 105
3.5.1 Contact force convergence under pointshell refinement . . . 107
3.5.2 Our contact model as a volume integral . . . 108
3.6 Sampling reduced deformable models . . . 109
3.6.1 Computing deformed point positions . . . 109
3.6.2 Computing deformed pointshell normals . . . 110
3.6.3 Alternative approaches to determining the contact normal . . . 112
3.7 Point-based BD-Tree . . . 114
3.8 Multi-resolution nested pointshell . . . 117
3.8.1 Multi-resolution pointshell via particle repulsion . . . 118
3.8.2 Pointshell generation alternatives . . . 121
3.9 Nested point-tree construction . . . 123
3.9.1 Nested point-tree construction for alternative pointshell generation approaches 125

3.10 Time-critical progressive contact forces . . . 125
3.10.1 Runtime tree traversal . . . 125
3.10.2 Temporal coherence . . . 127
3.10.3 Graceful degradation . . . 127
3.11 Rigid distance field . . . 130
3.11.1 Analytical implicit functions instead of a distance field . . . 132
3.12 Deformed distance field . . . 132
3.12.1 Multiple domains . . . 136
3.13 Experiments and results . . . 137
3.14 6-DOF haptic demo . . . 141
3.15 Discussion . . . 141

4 Haptic display of distributed contact **147**
4.1 The contact force and torque . . . 151
4.2 The contact force and torque gradients . . . 152
4.3 The virtual coupling force and torque . . . 154
4.4 The virtual coupling force and torque gradients . . . 156
4.5 Separating the simulation from haptics . . . 158
4.6 Limiting maximum speed of the haptic object . . . 159
4.7 Controlling maximum stiffness . . . 160
4.8 Static damping . . . 164
4.9 Virtual coupling saturation . . . 164
4.10 How to handle degenerate virtual coupling gradients . . . 167

Conclusion **171**

Bibliography **173**

List of Figures

1.1 **Deformable contact at high update rates:** Left: the two objects with complex geometry: a bridge (59,630 triangles), and a dinosaur (56,192 triangles). Bridge is deformable. Right: the user manipulates the dinosaur with a haptic device, feeling the contact forces and torques as the dinosaur enters contact with the bridge. The simulation computes deformations, detects collision and computes contact forces and torques in one loop running at 1000 frames per second. 21

1.2 **Large-deformation dynamics at kilohertz rates** are a key ingredient of deformable haptic simulations. Note the large bridge deformation resulting from contact with a user-controlled ball. Bridge is simulated in a 15-dimensional basis of deformations. Each timestep of the deformable object simulation took 65 microseconds to compute on a 3.0 Ghz Intel Pentium Xeon workstation with 2Gb of memory (manufactured in 2004). 24

1.3 **Distributed contact:** Left: Deformable dragon (deformable pointshell) in contact with a deformable dinosaur (deformable distance field). Right: haptic gaming: attacking the detailed rigid knight with a rigid axe. 28

1.4 **Real-time 6-DOF haptic rendering** of detailed deformable and rigid objects in contact. The operator is navigating the green rigid part in between two deformable hoses. 29

1.5 **Reduced simulations:** Left: Model reduction enables interactive simulations of nonlinear deformable models. Right: reduction also enables non-interactive large-scale multibody dynamics simulations, with nonlinear deformable objects undergoing free flight motion. The simulation is significantly faster than any other known approach to simulate such systems. Collision detection among the 512 baskets was resolved using BD-Trees [JP04]. Full video is available on the "StVK" project webpage [JJ]. 30

2.1 **A deflected cantilever beam:** Left: the deformed shape, Middle: strain, Right: stress. Both strain and stress are rank-2 tensors (3x3 matrices for all practical purposes in this thesis); the plotted color gives the maximum eigenvalue of the 3x3 matrix. 36

2.2 **Nonlinear FEM simulation of a voxelized elastic spoon:** The spoon consists of 2005 elements (voxels in this case) and has 11,094 degrees of freedom. Motion was computed using an unreduced approach: direct internal force and stiffness matrix evaluation, combined with the implicit Newmark timestepping scheme and a conjugate gradient solver. Spoon is constrained at the far end (blue), and the motion is induced by a short initial vertical impulse (green). Left-most picture is the rest pose, followed by three representative frames of the dynamic simulation (free vibration). 41

2.3 **Accuracy experiment:** Vertical displacement of a spoon simulation mesh vertex, located centrally at the tip of the spoon. Some deformations from this experiment were previously shown in Figure 2.2; the plotted simulation mesh vertex is colored green in Figure 2.2. Length of spoon is about 2.5 units. Triangle mesh poses are shown for reference. In all cases, motion is excited by the same identical initial short vertical impulse. The "basis from full motion" curve is obtained by performing Principal Component Analysis (PCA) on the motion computed by the full unreduced $11,094$-dimensional simulation, and then re-simulating the system in this basis. Note that in this case the basis is close to optimal for the simulation, and as such the curve almost aligns with the full simulation curve. This basis, however, wouldn't generalize well to other force loadings. "Modal derivatives" and "sketching" are two novel methods to generate the motion subspace, presented in our paper [BJ05]. The method of modal derivatives is automatic and makes no assumptions on the particular force loadings of the structure. All the reduced spaces are 12-dimensional. Our webpage [JJ] contains a video comparison corresponding to this Figure. 46

2.4 **Multi-resolution approaches to simulating deformable objects:** Left: an non-nested multiresolution hierarchy approach of [DDCB01], Right: and a subdivision basis approach of [CGC$^+$02a]. 49

2.5 **Subspace integration of Eiffel tower and heart models** 54

2.6 **Simulation Meshes:** Blue vertices are constrained. 56

2.7 **Geometric parameters of models from Figure 2.6.** 56

2.8 **Precomputing polynomial coefficients:** Reported numbers are totals for both reduced force and reduced stiffness matrix, and refer to single-processor computation. 59

2.9 **Runtime Computation Performance:** Integration times refer to one integration step. The number of integration steps per graphics frame is N. Timings for the heart and tower models are almost identical due to equal r. Machine used: Intel Pentium 3.0 Ghz processor (manufactured in 2004) with 2Gb RAM. 60

2.10 **Mass matrix Cholesky decomposition (bridge model, 11,829 simulation vertices):** Top-left: non-zero entries in the original mass matrix M (mesh generated via squashing cubes); Top-right: Cholesky factor of M; Bottom-Left: Permuted mass matrix (minimum degree ordering permutation); Bottom-Right: Cholesky factor of the permuted mass matrix. 65

2.11 **Basis from Sketch:** Left: User interacts with a linear model. Resulting shape is distorted. Center: Applied force is recorded and sent to an unreduced offline static solver to solve for the corresponding nonlinear shape. Several such shapes are then processed by mass-PCA to obtain a basis of motion. Right: If same force is re-applied during the reduced runtime simulation, a shape which is visually almost indistinguishable from the center image emerges. 66

2.12 **Precomputation Timings for the Basis from Sketch.** 66

2.13 **Dominant linear modes and modal derivatives:** We exploit the statistical redundancy of these modes using mass-PCA of suitably scaled modes. All vectors are shown mass-normalized. 69

2.14 **Extreme shapes captured by modal derivatives:** Although modal derivative are computed about the rest pose, their deformation subspace contains sufficient nonlinear content to describe large deformations. Left: Spoon ($k = 6, r = 15$) is constrained at far end. Right: Beam ($r = 5$, twist angle=270°) is simulated in a subspace spanned by "twist" linear modes and their derivatives $\Psi^4, \Psi^9, \Phi^{44}, \Phi^{49}, \Phi^{99}$. 70

2.15 **Computation of Modal Derivatives:** All performance data is given for a single 3.0 Ghz Pentium workstation with 2Gb of memory (manufactured in 2004). Mass-normalization and mass-PCA times were only a few seconds. 70

2.16 **Multibody dynamics simulation with large deformations:** In this simulation, we dropped 512 flexible baskets to the ground. The baskets started from zero velocity, horizontally separated (but in close proximity) at a certain height. Collision detection was performed using a BD-Tree [JP04]. Penalty forces were applied between leaf spheres to resolve contact. Motion basis ($r = 40$) used linear modes $\Psi_7, \ldots \Psi_{26}$ and their derivatives. The rigid body motion was not coupled with deformations, although this would not be a difficult extension [Sha05]. It took 1.2 sec total to perform the 512 implicit Newmark steps to advance the reduced deformations to the next graphical frame. 75

2.17 **The dragon model:** Left: the rendered dragon model in the rest configuration. Right: simulation voxel mesh. Bottom vertices are fixed rigidly. 84

2.18 **Dragon "head" force load:** Dragon dynamic deformations, originating from a short vertical force on a central vertex at the tip of dragon's head. Each row gives deformations under one method, with columns corresponding to timesteps $0, 5, 10, 15, 20$, with a timestep of $1/300s$ (see Figure 2.19). All reduced simulations use $k = 20$. 85

2.19 **Trajectory of a selected vertex:** Vertical displacement of simulation vertex where the force load was applied in the "head" sequence (Figure 2.18). 86

2.20 **Dragon "back" force load:** Dragon dynamic deformations, originating from a short force (horizontal, perpendicular to dragon's main axis) on a central vertex at the top of dragon's back. Each row gives deformation under one method, with columns corresponding to timesteps $0, 20, 40, 60$ (see Figure 2.21). All reduced simulations use $k = 20$. 90

2.21 **Trajectory of a selected vertex:** Z-displacement (i.e., horizontal and perpendicular to main dragon axis) of simulation vertex where force load was applied in the "back" sequence (Figure 2.20). 91

3.1 **Distributed deformable contact:** Top Left: contact configuration between a nonlinear reduced deformable bridge and a rigid dinosaur, with 12 separate contact sites. Other images: contact sites viewed from three different camera angles. A total of 5200 points were traversed, resulting in 426 points (in red) in contact. Points traversed but not in contact are shown in blue. Note: temporal coherence was disabled for this figure. 94

3.2 **VPS pointshell:** Left: the pointshell points for the teapot model. Right: a schematic representation of the pointshell, with inward normals. Both images are from [MPT99]. 95

3.3 **VPS voxmap:** Left: a VPS environment (a scene from the interior of an airplane, serving as source data for voxmap). Right: a schematic illustration of the voxmap, also showing the voxmap octree. Left image is from [MPT99]. Right image is an enhanced version of an image from [MPT99]. 96

3.4 **VPS in action:** Left: the user manipulates the teapot in the virtual environment. Right: the virtual coupling between the haptic device and the manipulated simulation object in contact. Both images are from [MPT99]. 97

3.5 **Virtual coupling:** Left: the green mechanical component (distance field haptic object) and the deformable hose scene (pointshell, with points shown in red). Wireframe shows the manipulandum position, solid green color shows the simulation object position. Right: a schematic representation of the manipulandum position (stippled black line) and simulation position (solid green line), together with the two coordinate systems which the virtual coupling tries to align. 99

3.6 **Point-contact model:** Left: Pointshell with inward normals. Middle: the signed distance field. Right: deformable contact and contact forces. 106

3.7 **Dynamic normals** (computed using our method): Left: undeformed. Middle, Right: deformed, viewed from two camera angles. In practice, normals need only be computed for pointshell points in contact. Maximum deviation from the correct surface normal is 5.2 degrees in this example; and less than 3.6 degrees for all but five points. 110

3.8 **(Inward) normals deformed with deformation gradients:** Left: undeformed configuration. Right: deformed configuration. Our pointshell surfaces are embedded into FEM volumetric meshes (voxel volumetric mesh is shown in gray). Volumetric mesh deformations drive the pointshell point positions and normals. . . 111

3.9 **Normals obtained only from the chair geometry** will give contact forces in the wrong direction for one of the two chair orientations. 114

3.10 **BD-Tree:** Left: The BD-Tree for the green arc geometry in the rest pose. Center: BD-Tree refitted under deformation. Right: the tree topology (which is never updated in the BD-Tree method). In our work, the geometry covered by the spheres are not arcs (as in this illustrative example), but pointshell points. 115

3.11 **Point-based BD-Tree:** Top row: the rest configuration. Bottom row: tree during contact. Right column shows the deepest bounding spheres that were traversed during this particular haptic cycle (i.e., each rendered sphere is free of contact). Note how the tree is traversed to different depths depending on proximity to contact. We typically use many spheres (320 in this example) on the first level, followed by 4x more spheres on consecutive levels. First-level spheres are always traversed (therefore they appear in top-right image) if temporal coherence is off; however, when temporal coherence is enabled most of them need not actually be checked during most haptic cycles. There are 4 hierarchy levels in this example. The small spheres next to the green metallic part in the bottom-right image are level 3 spheres. Level 4 spheres have zero radius, as they correspond to leaf points. 116

3.12 **Point-based BD-Tree performance under large deformations:** The sphere radii do grow under large deformations (see, e.g., two level-1 spheres at the tip of the upper u-shaped hose), however tree traversal is still very efficient. The upper hose is bent at an angle of approximately 25 degrees. If the green part is lowered (but kept inserted in between hoses) such that the upper hose returns to (almost) rest configuration, the performance changes from $220\mu s$ to $210\mu s$. To the best of our knowledge, while limited to reduced deformations (as in our case), BD-Tree is the only deformable bounding volume hierarchy update method that can reach haptic update rates (1000 Hz) with example complexity similar to ours. In general, the BD-Tree performs especially well when large deformations are localized to a part of the scene; also, BD-Tree could be augmented by computing bounds at configurations other than the rest configuration. 117

3.13 **Unstable pointshell:** This "nightmare" pointshell will result in a very unstable (chaotic) haptic signal. 118

3.14 **Multi-resolution oriented pointshell from particle repulsion:** Left: The first two levels of a part of the hose scene. Right: detail with four levels shown. Points are fitted on a small-distance offset oriented manifold surface to support "polygon soup" input geometry, and provide oriented surface normals for contact. 120

3.15 **Pointshell for thin structures (e.g., cables):** First and third image from left: the bridge geometry, viewed from two camera angles. Second and fourth image: the pointshell. Each red dot is one point. Points are fitted onto a level set (offset surface) of an unsigned distance field to the bridge geometry. Distance field resolution was 256x256x256 for this pointshell, and offset distance was 1.4x the size of a distance field voxel. 121

3.16 **Pointshell for thin structures with a smaller offset:** Left: pointshell (four levels, 85,000 points) fitted using a 256x256x256 unsigned distance field (same pointshell as in Figure 3.15). Right: pointshell (five levels, 256,000 points) fitted using a 512x512x512 unsigned distance field, and with a 2x smaller offset distance as pointshell on the left. Both pointshells use an offset of 1.4x the size of the distance field voxel. Note that the finer pointshell resolves all twelve central vertical cables on the bridge, whereas the coarser pointshell merges groups of three cables together. Simulation with the finer pointshell is more permissive (accurate) in that thicker objects can be inserted through holes in pointshell geometry, such as the holes in the bridge's upper shelf. In general, higher resolution unsigned distance fields are required for smaller offsets, otherwise the offset surface becomes progressively non-smooth (noisy). 122

3.17 A point cloud generated using the Layered Depth Cube algorithm. 123

3.18 **Nested point-tree:** Left: The pointshell. Right: the hierarchy, the traversal order and tree levels $\mathcal{L}_i$. Particle-repulsion levels are 0-1, 2-5, and 6-19 in this case. . . . 124

3.19 **Tree traversal:** Left: The bounding sphere of node 0 intersects the environment and all the children are queued for traversal (indicated by the symbol 'Q'). Right: Node 1 is sufficiently far from contact that its bounding sphere does not intersect the environment, and as such, the subtree need not be traversed further. This completes the traversal of Level 1. The traversal list for Level 2 is $\{0,\ 4,\ 5\}$, where 0 is in this case the copy on level 2. 126

3.20 **Graceful degradation:** Left, Middle: With involved contact scenarios, many tree nodes need to be traversed during a complete hierarchy traversal. Right: Under graceful degradation, tree is traversed only up to a shallow depth, as permitted by available CPU power. 128

3.21 **The proof of Inequality 3.14.** 130

3.22 **Haptic contact between reduced deformable objects and rigid objects modeled by simple analytical functions:** Left: The user-manipulated deformable chair is a deformable pointshell with about 26,000 vertices, rendered against an analytical implicit function of two perpendicular walls. Right: The deformable bridge is a deformable pointshell of about 41,000 vertices, rendered against a user-manipulated sphere. The chair is a linear reduced deformable model (linear modal analysis deformations). The bridge is a nonlinear StVK reduced deformable model, as described in the first part of the thesis. 133

3.23 **Deformed distance field approximation:** Left: Proxies (squares), the query pointshell point at x. Middle: 3-nearest neighbors and their weights. Right: material configuration pull-backs X_i of x. 135

3.24 **Isocontours of the exact and approximate deformed distance field:** Left: Pose 1, single-domain approximation. Middle: Pose 2, single-domain approximation with the two legs mirroring. Right: Pose 2, multiple domain approximation without mirroring. Green denotes the interior of the object, with respect to the approximated field. 136

3.25 **6-DOF haptic rendering:** Left: the user is manipulating a rigid Buddha (distance field object) against the deformable dragon (pointshell object). Right: the user is solving the alpha puzzle, while the device provides contact force and torque feedback (see also Figure 3.31). The device in the images is a Phantom Premium 1.5 6-DOF device from Sensable Technologies. 138

3.26 **Our method can render small features:** Left (top and bottom): resolution in our method is high enough for the user to feel the bumps on the back of the rigid dinosaur (distance field object, 256x256x256), by sliding the back of the dinosaur against the upper shelf of the bridge (pointshell object). The red arrows are indicating the direction of the dinosaur's sliding. Both objects are rigid in this experiment. Bridge pointshell has five levels of detail with 256,000 points (same L5 pointshell as in Figure 3.16, Right, on page 122). Right: the force magnitude rendered to the user, under progressively finer pointshells and a pre-recorded manipulandum trajectory. Sampling rate is 1000 frames per second. The user caused six bumps total to touch the bridge (one after another, in tangential motion). The impact of each individual bump can be clearly seen in the haptic signal, and felt in the haptic simulation. 139

3.27 **Illustration of graceful degradation (GD):** Simulation data for a pre-recorded manipulandum trajectory is rendered in two ways: using GD (in red) with warm threshold at 600 nodes; without GD (in black) with all four tree levels. The pointshell has $85,018$ points total (same L4 pointshell as in Figure 3.15 on page 121). The common x-axis corresponds to haptic frames. This data was recorded on a three-year-old 3.0 GHz dual Intel Xeon processor with 2.75 Gb of memory, illustrating algorithm's adaptivity to different computer speeds. On the machine of Table 3.1, the black timing curve reaches a maximum of $\sim$800 μs. 140

3.28 **Graceful degradation supports large conforming contacts** to allow time-critical approximation of very difficult contact configurations (here under 1 millisecond). Top Left: A dragon "peg" inserting into a matching dragon "hole" obtained via CSG difference. Top Middle: dragon manipulated haptically into the hole. Top Right: Large contact area with traversed points in blue (43% of all L2 points), and contacting points in red (21% of all L2 points). (**Bottom Graphs**) Simulation data for a pre-recorded manipulandum trajectory is rendered in two ways: using GD (in red) with warm threshold at 2000 nodes; without GD (in black) with all five tree levels. Same machine as in Table 3.1. The common x-axis corresponds to haptic frames. 144

3.29 **Asymmetric contact model:** The contact force changes under role reversal, however the error is quite reasonable. The first contact case (green dragon=pointshell, brown dragon=signed distance field) is depicted with a solid red line, the reverse case (green dragon=signed distance field, brown dragon=pointshell) is shown in dashed blue. Both lines show the raw contact force on the green (i.e., moving) dragon. The top-right image shows the configurations at samples 900 and 1800, respectively. The bottom-right images depict points in contact (red=in contact, blue=traversed by tree, but not in contact). Note that the z-axis is aligned with the direction of motion, and is (about) normal to the contact areas; z-force is therefore the normal contact force and is as such substantially larger than tangential forces. Penetration depths are large in this experiment; in our simulations, virtual coupling coupling saturation would prevent penetrations deeper than about 1 voxel. The vertical green line denotes the sample index where penetration reaches 1 voxel. 145

3.30 **Asymmetric contact (detailed plot for shallow penetrations):** the contact forces are not symmetric, however the difference is small for shallow penetrations. Same notation as in Figure 3.29. X-axis corresponds to sample indices. 146

3.31 **Interactive path planning with 6-DOF haptics (Alpha puzzle):** The goal is to position the rigid blue alpha-shaped tube inside the loop of the rigid red alpha-shaped tube. The red and blue tubes have identical shapes. The red tube (pointshell object, 5571 points, 4 levels total) is fixed in space, and the blue tube (distance field, 128x128x128) is the haptic object. The gap between the two sides of each alpha (best seen with the blue alpha in the middle image) is too small for trivial attempts to succeed, both in the real-world and in our haptic simulation. With careful manipulation, the two solutions to the puzzle can be discovered. Left: initial configuration (106μs of computation per haptic cycle, 2 tree nodes traversed, 0 points in contact). Middle: a configuration in the middle of one of the two solution paths (156μs, 233, 5). Right: puzzle solved (108μs, 16, 0). Path planning is accurate up to the penetrations permitted by maximum contact forces under virtual coupling saturation, e.g. typically one half of a distance field voxel. Also note that we could make this example deformable, but then the solution would be very easy. 146

4.1 **Grasping location affects virtual coupling forces and torques:** The user is manipulating this long pole against a rigid obstacle (a cylinder viewed from top), and will feel different forces and torques depending on the choice of X_{grasp} (Left: at pole center, Right: at pole far end). Stippled line denotes the manipulandum and solid line denotes the simulation object. Virtual coupling tries to align the two coordinate systems. 148
4.2 **Block diagram of one haptic cycle:** SPO=Simulation object Position and Orientation, MPO=Manipulandum Position and Orientation. 150
4.3 **The contact model** 151
4.4 **Multiplane penalty contact** ($\ell = 3$). 161
4.5 **Virtual coupling torque saturation:** Left: without torque saturation, the user can squeeze the blue alpha in between the other (red) alpha's gap, by using the blue alpha's long end as a lever. Right: A fixed torque saturation limit will overpower contact penalty forces for long objects, such as for this manipulated long (horizontally positioned) rod in contact with a static vertical cylinder (view from top). 167

List of Tables

2.1 **Reduced systems have high frequency spectra:** Linear spectrum row gives the range of the first 20 natural linear modal frequencies of the structure. For both the Eiffel tower and the heart model, the Young's modulus was chosen such that the lowest frequency is 1 Hz, so that frequencies are in an interesting range for computer graphics; but the same properties carry over to stiffer structures with higher frequency ranges. The reduced spectrum row gives the range of the frequencies of the linearized *reduced* system, i.e., the range of $\sqrt{\lambda}/(2\pi)$, where λ traverses the eigenvalues of the reduced tangent stiffness matrix $\tilde{K}$ at the origin. The reduced system was obtained by augmenting the first $k = 20$ linear modes with their derivatives, and mass-PCAing this assembly of vectors to a $r = 30$ dimensional basis. For comparison, we also give the spectrum of the unreduced FEM model (last row). 78

3.1 **Model statistics** for selected contact configurations, timed on an Intel Core 2 Duo 2.66 GHz processor with 2 Gb RAM; r=rigid, d=deformable, VC=virtual coupling and manipulandum transformations, StVK=deformable FEM dynamics, TT=tree traversal, TC=temporal coherence, LOD=level of detail rendered (in the particular configuration), TL=time under a tree-less pointshell traversal. All deformable models use $r = 15$. 137

Chapter 1

Introduction

This thesis is about real-time simulation, an interesting inter-disciplinary area spanning computer graphics, applied mathematics and physics. The problem investigated is the following: given two (or several) 3D objects, specified, say, by providing surface triangle meshes for each object (see Figure 1.1, left), how can we build a *real-time* computer simulation where the user can interactively manipulate one of the objects (dinosaur in Figure 1.1), potentially entering contact with the other object (bridge in Figure 1.1). The simulation has to check for collisions between objects at real-time rates, and compute and apply appropriate contact forces to push them apart if in contact. Furthermore, the simulation should support objects deforming due to contact forces.

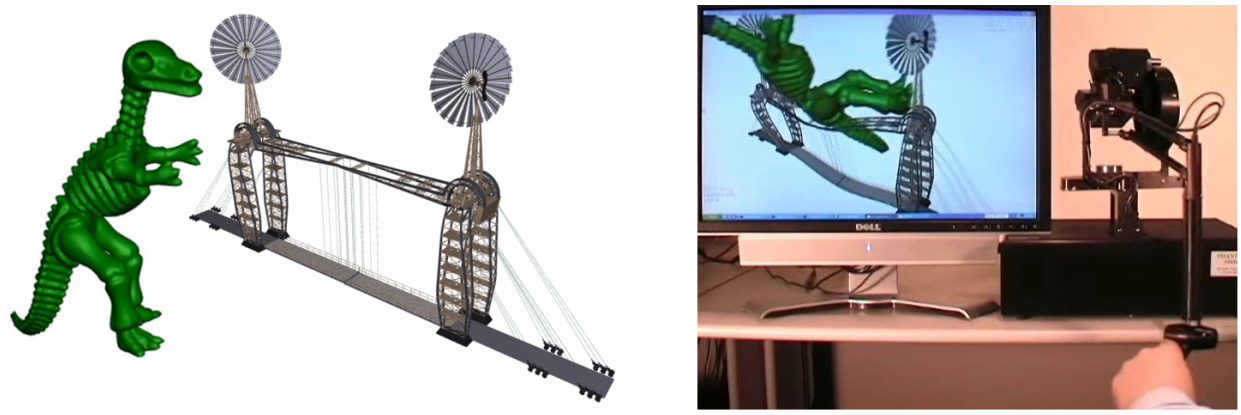


Figure 1.1: **Deformable contact at high update rates:** Left: the two objects with complex geometry: a bridge (59,630 triangles), and a dinosaur (56,192 triangles). Bridge is deformable. Right: the user manipulates the dinosaur with a haptic device, feeling the contact forces and torques as the dinosaur enters contact with the bridge. The simulation computes deformations, detects collision and computes contact forces and torques in one loop running at 1000 frames per second.

Interactive simulations are not convincing to the user unless they are performed at sufficiently high rates. If the positions, orientations and deformations of objects on the screen are updated with

less than about 30 times per second, users will no longer perceive the simulation as continuous motion. The requirements are even more stringent if the simulation also displays the contact forces and torques using a *force-feedback device* (see Figure 1.1, right). Typically, 1000 steps per second are required in order to keep the haptic controller stable and maintain the illusion of virtual touch.

For complex geometry (especially if deformable), collision detection and contact handling are computationally intensive operations, which makes it difficult to maintain such high real-time update rates. Two 3D objects can enter contact in a variety of contact configurations: single point-like contact, a single contact with a wide contact area, or there can be several simultaneous contact sites, each distributed over a certain surface area. The contact force algorithm has to be both fast and robust, producing stable contact forces for all contact configurations. Furthermore, if objects are deformable, it is also necessary to simultaneously compute the deformations of the objects, which is a computationally intensive operation on its own (for objects with complex geometry). Collision detection also becomes more difficult in the presence of deformations. This thesis presents a set of algorithms which enable such interactive simulations of general 3D objects, in a way that scales to complex geometry. We make certain assumptions that make the problem tractable, such as assuming a penalty contact model and simplifying deformable object simulation using model reduction. Our deformable objects are therefore not fully general; they are, however, fast and support large deformation dynamics. To the best of our knowledge, this thesis is first to present deformable contact at haptic rates (1000 simulation steps per second) between two objects, in a way that scales to both objects having complex geometry.

The first part of the thesis is about fast deformable object simulation. We present a class of dimensionally reduced nonlinear deformable objects which can be timestepped at very high rates, trading simulation accuracy for speed. In physics, deformations of solid objects are governed by the partial differential equations of continuum mechanics. These equations do not have tractable analytical solutions for all but the simplest geometries. For complex geometries, equations can be solved numerically, such as, for example, by using the Finite Element Method (FEM). FEM is the most commonly used discretization approach for deformable object simulation (with complex geometry) in applied mathematics and engineering, and it is widely considered to give good quality simulations with sufficiently fine meshes. Fine meshes, however, result in a large number of deformable degrees of freedom. Furthermore, the partial differential equations that describe *large* deformations of objects are nonlinear. FEM therefore produces high-dimensional nonlinear equations of motion which describe the deformations very accurately, but require too much computational effort for simulations on commodity hardware at high update rates. Instead, we simplify these general FEM models using dimensional model reduction, obtaining *reduced deformable models*. These models are nonlinear and support *large* deformations and deformable dynamics, both of which play an important part of computer graphics and animation. Reduced deformable models can be timestepped with little computational effort, but are not as accurate as the general FEM models. The deformable objects in this thesis are three-dimensional, volumetric, and assumed to be elastic: they return to the original rest shape if external forces (loadings) are removed. We do not investigate plastic deformations and fracture.

In the second part of the thesis, we present an algorithm to simulate two 3D reduced-deformable

(or rigid) objects in contact. We address three contact cases: rigid vs rigid, reduced-deformable vs rigid, and reduced-deformable vs reduced-deformable contact. We combine this contact algorithm with the reduced deformable model from the first part of the thesis to build 6-DOF haptic simulations of contact between two 3D reduced-deformable objects. Our algorithms scale to complex geometry. We simulate *distributed contact*, i.e., contact with (potentially) several simultaneous contact sites, each distributed over a non-zero surface area. Note that distributed contact is the most general contact between two 3D objects. We resolve distributed contact using a CPU-based algorithm which uses a (deformable) multi-resolution point cloud (the *pointshell*) for one object and a (deformable) signed distance field for the other object. Reduced-deformed distance fields are proposed to support contact between reduced-deformable objects. We resolve contact by applying penalty forces and torques to points which penetrated the zero isocontour of the signed distance field of the other object. We investigated several possibilities for the particular contact model, and eventually selected a contact model novel to haptic rendering which is (1) simple to evaluate, (2) gives stable haptic contact forces, and (3) converges to a well-defined limit under progressively finer pointshells. Furthermore, we organize the pointshell into a *nested point-tree*, a novel nested multi-resolution datastructure which can be used to interrupt the contact force computation in order to maintain haptic update rates. Nested point-tree supports graceful degradation through a progressive multi-resolution construction of contact forces. The accuracy of the contact forces computed within the 1 msec haptic frame is adapted to both the speed of the particular computer, and the difficulty of the current contact configuration. We also demonstrate how to exploit temporal coherence to accelerate the simulation. We present several examples of 6-DOF haptic rendering of geometrically complex rigid and/or deformable objects in contact, including simulation of aircraft geometry, interactive path planning and game haptics.

1.1 Fast deformable models using model reduction

Researchers in computational physics, mechanics, and applied mathematics have been developing numerical algorithms for simulations of deformable objects for over 50 years. However, partial differential equations of solid continuum mechanics are demanding and permit few computational shortcuts. This effectively prohibits fully general *interactive* simulations of deformable objects with detailed geometry and non-trivial material properties. Namely, in such simulations it is necessary to update the state of the system at very fast rates, such as (for example) 30 times per second for quality visual feedback and 1000 times per second for force feedback. In the year 2007, it typically takes at least one second for a fully general deformable method to compute one timestep of a nonlinear FEM simulation of a detailed model, such as a liver model with 15,000 hexahedral elements. A nonlinear FEM multigrid algorithm can simulate meshes with about 1500 tetrahedra at update rates of about 20 times per second on commodity PC hardware [GW06]. Also, in 2004 we have personally implemented the corotational deformable object approach of [MG04], and achieved a performance of about 30 frames per second for meshes with about 1000 tetrahedra.

Objects undergoing physically based *large* deformations play an important part of computer

graphics and animation. For real-time applications low-latency solutions are required, and a trade-off is typically made between physical accuracy and speed. For example, the high update rates of force-feedback haptic rendering make it difficult to simulate large deformations accurately, especially with complex geometry and distributed contact interactions (see Figure 1.2). Many interactive and offline simulations, such as those used in the computer animation industry, might also benefit from having highly interactive large-deformation models.

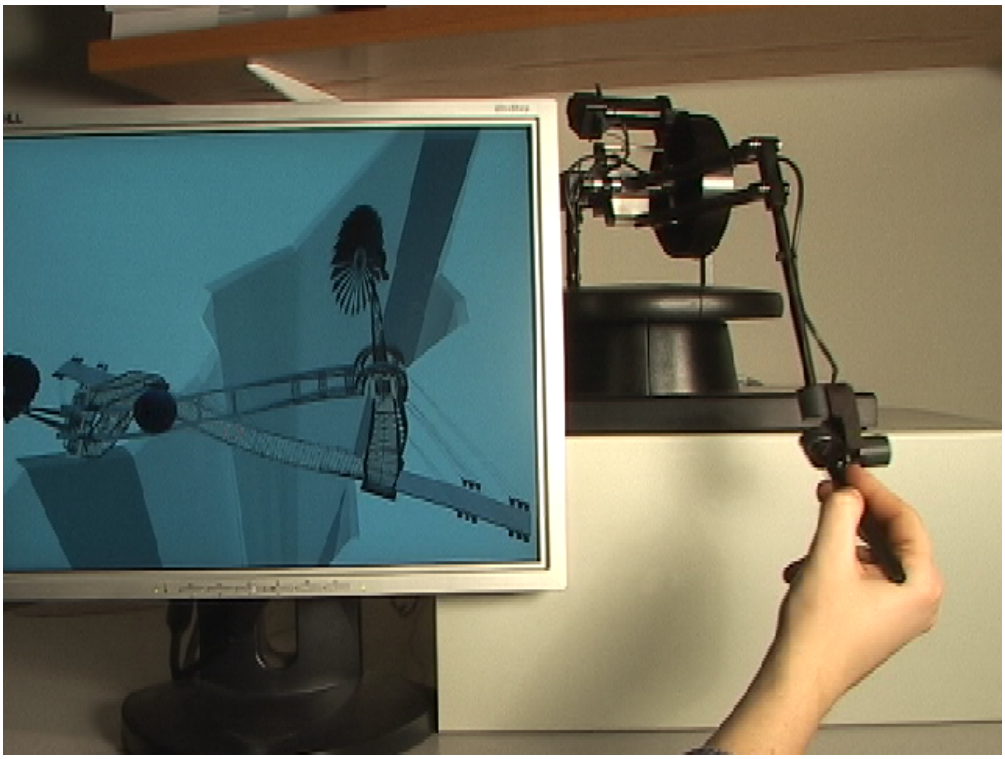


Figure 1.2: **Large-deformation dynamics at kilohertz rates** are a key ingredient of deformable haptic simulations. Note the large bridge deformation resulting from contact with a user-controlled ball. Bridge is simulated in a 15-dimensional basis of deformations. Each timestep of the deformable object simulation took 65 microseconds to compute on a 3.0 Ghz Intel Pentium Xeon workstation with 2Gb of memory (manufactured in 2004).

The interactive deformable object problem has been a subject of much research in computer graphics. Obviously, to be able to reach high update rates, the methods cannot achieve the same accuracy as offline solid mechanics solvers. To gain speed, one simplification has been to use some variant of the linearized equations of motion [BNC96, CDA99, JP99, JP03]. Existing interactive *nonlinear* FEM approaches include "stiffness warping" corotational approaches [MDM$^+$02, MG04], and multi-resolution techniques [DDCB01, CGC$^+$02a, GKS02]. The latter adaptively change the simulation basis to accommodate runtime detail.

We propose an alternative approach: interactive nonlinear deformable object simulation using model reduction. These deformable models give a compact representation for large global

structural vibrations of deformable objects, resulting in very fast update rates. Nonlinear reduced deformable models are a previously unexplored area of interactive computer graphics, and the resulting algorithms are faster than any other known method for a significant class of objects. For example, large structural vibrations of buildings, bridges, trees, airplane mechanical components, can be efficiently modeled by a small set of reduced degrees of freedom. Applications are possible in interactive virtual medicine to model large-scale human tissue deformations. Inexpensive deformable objects are also useful in computer games, where only a small fraction of the CPU can be devoted to deformations, as several subsystems must all run together at real-time rates: artificial intelligence, rendering, sound, game physics, network protocol if a multiplayer game, etc. For example, reduced deformations could be used to model the motion of plants in the wind, or a rope bridge interactively swaying under the feet of a game character.

The reduced nonlinear deformable approach presented in this thesis works as follows. During pre-process, model reduction is used to automatically identify a quality *low-dimensional* space of "typical" deformations. The equations of motion are then projected onto this motion space. At runtime, a reduced ODE solver can timestep the solution much faster (with some loss of accuracy) compared to a fully general unreduced deformable solver. Such fast simulations are possible especially in the case of *linear stress-strain material laws* (note that the overall simulation is still nonlinear due to nonlinear strain), because in this case, the reduced internal forces turn out to be cubic polynomials in the reduced coordinates, and can be precomputed. This makes it possible to build hard real-time reduced simulations of large-deformation dynamics. For example, the simulation speedup (of course, with some loss of simulation accuracy) was about 50,000x when we reduced an elastic spoon with 11094 DOFs to $r = 12$ reduced degrees of freedom. With $r = 30$ reduced degrees of freedom the speedups in our examples were on the order of 10,000x. Reduced *static* simulations are also possible if desired; simply discard the mass-inertia and damping terms in the reduced equations of motion. Linear material laws are *anisotropic* in general, meaning that the material is more compliant ("stretchable") in certain directions than others; but an important special case is the well-known *isotropic* Saint-Venant Kirchhoff (StVK) material model, parameterized by the familiar parameters of Young's modulus and Poisson ratio. This material is very common in graphics: it is the simulation model in several important papers on (real-time) deformations in computer graphics (it is usually referred to as a *geometrically nonlinear* or *large-deformation model*). Under our reduction approach, all linear material laws are equally fast at runtime, so a general linear material model could be used, for example, to model anisotropic properties of human tissue.

Dimensional model reduction for nonlinear deformable object modeling has appeared in the mechanics literature before [KLM01], and this thesis extends it by providing an automatic motion subspace selection technique, and a fast reduced internal force evaluation algorithm for linear material models. Model reduction is possible because of the following observation: when complex objects deform, different parts of the model deform coherently. Think of a large bridge swaying in the wind. It usually does not just deform arbitrarily - there are some typical key shapes that it tends to deform into. When deformable body motion contains only a small amount of information, this presents an opportunity for fast numerical simulation. Individual vertices move in a

coherent way, and hence, statistically, the deformations lie in some characteristic low-dimensional linear subspace of all possible deformations. This makes it possible to substitute the full set of body's degrees of freedom with a much smaller set of degrees of freedom, which can be integrated much faster. In this thesis, we present an automatic process that can identify such typical shapes directly from the principles of continuum mechanics (*a priori* model reduction). Alternatively, it is also possible to use pre-existing simulation data to identify typical shapes using dimensionality reduction techniques, such as Principal Component Analysis (PCA) (*a posteriori* model reduction). Note that the problem of identifying typical deformations has already been addressed in the literature for the case of small deformations (this is *linear modal analysis*), where linear physics is sufficient. Our approach can extract a space of typical deformations for the more general case where deformations are allowed to be large, and where the nonlinear terms in the equations of motion are significant (see Figure 2.14). The automatic motion subspace selection technique can construct a meaningful motion basis for arbitrary 3D meshes and arbitrary physical material parameters (mass density, Young's modulus, Poisson ratio, or general linear material laws in case of anisotropy, or even nonlinear material properties). The material parameters can be inhomogeneous (vary across the object).

1.2 Real-time deformable contact and haptic rendering

The computation of deformations is just one of the challenges with interactive simulations of deformable objects. Collision detection and contact handling for deformable objects are crucial aspects of the problem too. This thesis presents a novel algorithm for haptic rendering of deformable objects, which can simulate contact between rigid and reduced deformable objects with complex geometry (including reduced-deformable vs reduced-deformable contact). The contact simulated is *distributed* (also sometimes called *multi-point*): this is contact between 3D objects with potentially several simultaneous contact sites, each distributed over a non-zero surface area. Distributed contact is the most general kind of a contact between two 3D objects, and represents an important component of interactive physically based virtual environments. Distributed contact simulations are challenging due to two competing demands: (1) having interesting geometrically complex rigid and (more notoriously) deformable objects, as well as (2) high real-time contact force calculations rates, e.g., 1 kHz to support haptic force-feedback rendering. A recent survey of haptics research laments: *"In the simulation of real environments, interactive geometric computations on complex models are a fundamental problem"* [Hol00a].

This thesis proposes an efficient CPU-based algorithm suitable for hard real-time evaluation of distributed contact forces between geometrically complex models. These real-time forces can be used in a variety of applications, however, we focus on the challenging (6-DOF) haptic rendering of deformable models. Our approach was motivated by the Voxmap-PointShell method (VPS) [MPT99], a 6-DOF haptic rendering algorithm capable of simulating distributed geometrically detailed contact between a *rigid* environment and a *rigid* point-sampled object (*pointshell*). The pointshell is a collection of points (with normals) positioned near-regularly on the surface of

the object, similar to surfels in point-based rendering [PZvBG00]. The VPS environment is modeled as a *voxmap* datastructure, a voxelization of the polygonal environment geometry where each voxel contains a fixed-width (e.g. 2-bit) value depending on its distance to the geometry. At every haptic cycle, pointshell points are queried against the voxmap to determine contact forces.

Unfortunately, objects modeled by VPS and notable recent works on 6-DOF haptic rendering are entirely rigid, and for good reason: nonlinear deformable objects are computationally expensive, and general FEM solid mechanics deformable object simulations do not even run at haptic rates (1000 Hz) for geometrically interesting models. Also, collision detection is much more expensive in the presence of deformations. Therefore it is difficult to build haptic simulations of detailed deformable models in detailed environments, and this situation severely limits many emerging applications. For example, haptic simulations are useful for virtual prototyping: airplane designers need to check if a certain mechanical part can be positioned into (or dismounted from) its designated location, to avoid design flaws without actually manufacturing the structure (e.g., fire detector removal from an aircraft engine during servicing [WM03]). Simulating deformable geometry is important: airplane geometry is often rigid, but hoses, wires, tubes, etc., are compliant. When rigid simulations are used to check for mechanical part removability, they can be unnecessarily restrictive and may report that a part cannot be (dis)assembled when in reality it can be. For example, wires and hoses may be deformed to let an object through a narrow passage (see Figure 1.4). Supporting deformations is therefore crucial in building such simulations, and we found the geometrically nonlinear reduced deformable models of this thesis to be a natural choice because of the balance between accuracy and speed. We also make extensive use of the low-dimensional reduced shape parameterization for time-critical collision and force processing.

We use a floating-point signed distance field instead of a voxmap. This will make the contact force continuous across voxel boundaries (improving stability), and also allows us to benefit from temporal coherence. A rigid distance field and a deformable pointshell make it possible to simulate haptic contact between a *detailed* rigid and geometrically *detailed* (reduced) deformable model for the first time. In addition, we present a novel extension that permits approximate evaluation of deforming distance fields for reduced deformable models, which ultimately enables haptic rendering of two detailed deformable objects in contact (see Figure 1.3).

To support real-time haptic display of geometrically detailed models, e.g., involving a million pointshell points, we resolve distributed contact hierarchically. We organize the point shell into a multi-resolution hierarchy, by sampling points and normals from an appropriately defined smooth manifold surface. Near uniform sampling also provides more economic pointshells with improved haptic stability. For rigid pointshells, a precomputed sphere-tree hierarchy is used to bound the locations of points in a subtree. For deformable models we introduce a point-based modification of the Bounded Deformation Tree (BD-Tree) [JP04] wherein sphere centers correspond to suitable pointshell points to enable time-critical evaluation of contact forces. Multi-resolution approaches permit large pointshells, bigger than what could be examined point by point during one haptic cycle: if a point on some coarse hierarchical level is sufficiently far from contact, none of the points in the subtree can be in contact, and then those points need not be checked. In contrast, if a large part of the pointshell is in contact or close to contact, few subtrees will be culled and a

Figure 1.3: **Distributed contact:** Left: Deformable dragon (deformable pointshell) in contact with a deformable dinosaur (deformable distance field). Right: haptic gaming: attacking the detailed rigid knight with a rigid axe.

large part of the pointshell will need to be examined point by point. We address this problem by providing *graceful degradation of contact*: if there is not enough computation time to fully complete the tree traversal, the algorithm still returns a reasonable answer, with accuracy dependent on contact-configuration difficulty and available processing power. Graceful degradation is achieved by traversing the hierarchy in a breadth-first order, rendering deeper and deeper tree levels until out of computation time. We use two separate activation thresholds to avoid abrupt changes in the rendered depth during consecutive haptic cycles. We also use temporal coherence to accelerate the simulation.

We achieve haptic display of distributed contact forces and torques using a novel first-order quasi-dynamic virtual coupling method that uses both forces and torques, and related rigid-motion derivatives. In haptics, higher update rates generally enable stable rendering of stiffer forces and torques; but reaching such high rates is challenging for models with detailed geometry. In our system, the deformations, collision detection and contact force computations run together in one loop at haptic rates (1000 Hz or greater). We do not need to extrapolate from lower-rate signals, although this could further increase example complexity [OL06].

1.3 Thesis overview

Chapter 2 is devoted to fast deformable models via model reduction. In Chapter 3, we describe our algorithm to compute contact forces and torques at high update rates. Chapter 2 stands on its own as a fast deformable object algorithm for use in computer animation. The algorithm of Chapter 3 uses

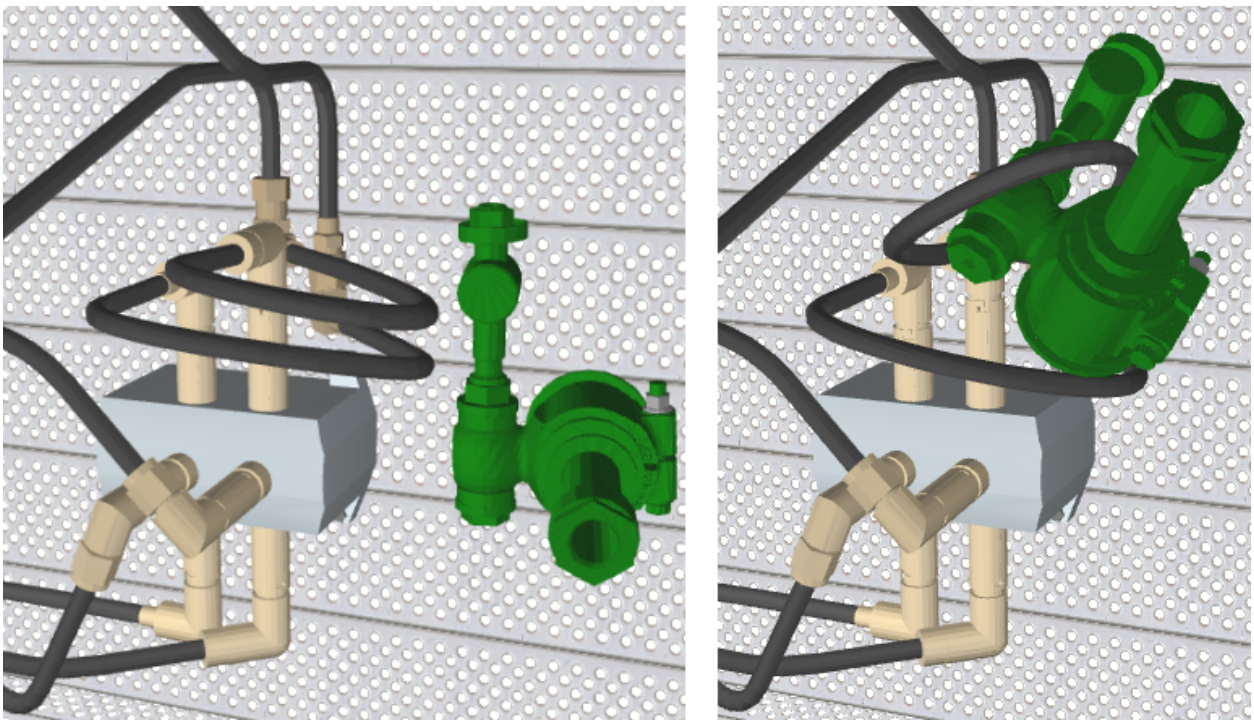


Figure 1.4: **Real-time 6-DOF haptic rendering** of detailed deformable and rigid objects in contact. The operator is navigating the green rigid part in between two deformable hoses.

the deformable models of Chapter 2. Other types of reduced models could be used instead, such as linear vibrational models, FFD models where a coarse mesh drives the deformations of a finer mesh, articulated models, or even multi-resolution models. Several research results of Chapter 3 apply both to rigid simulations and deformable simulations, such as our nested point-tree and graceful degradation algorithm. Chapter 3.4 describes all the necessary components to build a 6-DOF haptic rendering system. We demonstrate how to integrate the algorithms of Chapters 2 and 3 to achieve stable haptic rendering of contact between objects with complex (reduced-deformable) geometry. We give description of several subtle, but important details which make 6-DOF haptic rendering possible, and which were previously either not documented or only partially documented in haptic rendering literature.

1.4 Thesis contributions

The thesis has contributions to computer graphics, mechanics, and haptics.

Contributions to computer graphics: We give a new algorithm to simulate deformable objects at very high rates. The algorithm can be seen as an extension of linear modal analysis to large deformations. We show how modal derivatives form a natural correction to the artifacts of linear modal

analysis for large deformations. More broadly, the thesis introduces dimensional model reduction to physically based modeling. Our real-time solid mechanics approach [BJ05] has inspired real-time fluid simulations based on model reduction [TLP06]. We also give a GPU implementation to synthesize modal deformations in the pixel shader, where the number of reduced coordinates supported is limited only by the size of texture memory.

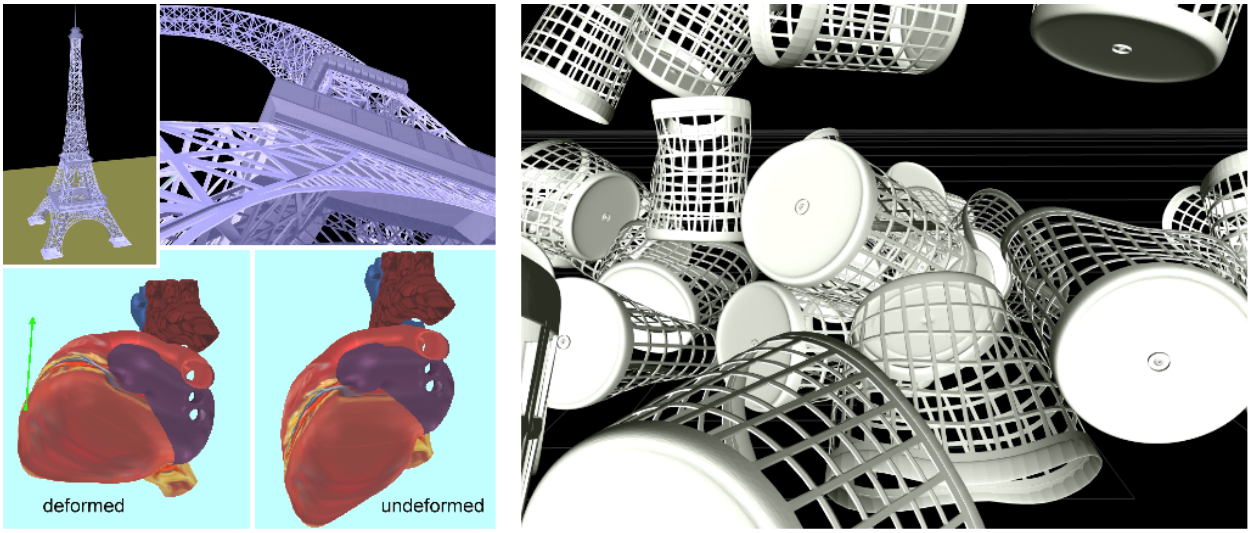


Figure 1.5: **Reduced simulations:** Left: Model reduction enables interactive simulations of non-linear deformable models. Right: reduction also enables non-interactive large-scale multibody dynamics simulations, with nonlinear deformable objects undergoing free flight motion. The simulation is significantly faster than any other known approach to simulate such systems. Collision detection among the 512 baskets was resolved using BD-Trees [JP04]. Full video is available on the "StVK" project webpage [JJ].

Contributions to mechanics: The first work in solid mechanics community to perform model reduction statistically was [KLM01]. These "empirical eigenvectors" were found by applying PCA to pre-existing deformation trajectories. This thesis goes beyond [KLM01] as follows:

- The "empirical eigenvectors" of [KLM01] require pre-simulation to obtain deformation data on which to perform model reduction. We provide an automatic basis generation technique, by performing PCA model reduction on linear modes and modal derivatives [IC85b] (a priori modal reduction). This requires no pre-existing simulation trajectories or advance specifications of force loads.

- For the important special case of geometric nonlinearities, we identify reduced internal forces as cubic polynomials, and give an algorithm to precompute their coefficients. This enables fast runtime evaluation of reduced internal forces and tangent stiffness matrices.

- If pre-simulation is desired, we show how it can be guided via a user's sketch using a linear model; this means that the user can essentially bootstrap from a linear model to a nonlinear model.
- We can also handle unconstrained objects, where large deformations are superimposed on top of rigid body motion (see the baskets in Figure 1.5). Such an extension is not obvious due to singular stiffness and other matrices that arise with deformable models possessing rigid degrees of freedom.
- We argue that using standard PCA (as in [KLM01]) for model reduction is not optimal. We introduce *mass-scaled* PCA, which, within the family of PCA-like algorithms, we claim to be the natural dimensionality reduction algorithm for solid mechanics problems.

Contributions to haptics: We demonstrate a system to perform (6-DOF) haptic rendering of *distributed* contact (i.e., potentially several contact sites with varying surface areas) between objects with rigid or reduced-deformable geometry. Our approach scales to allow both contacting objects to have complex geometry. To the best of our knowledge, the geometric complexity of our models exceeds anything previously demonstrated for haptic rendering of contact between deformable objects. We can also address the especially challenging haptic contact between two (reduced) deformable objects; we do so using a novel fast approximation of signed distance fields for reduced deformable objects. The approximation runs at haptic rates for several hundred signed distance field evaluations per each haptic cycle.

We achieve these haptic rendering results by presenting the following ideas, novel with respect to the state of the art of the Voxmap Pointshell method (VPS) at the time of writing this thesis (July 2007). We suggest using hierarchical particle repulsion to improve the VPS pointshell sampling, and using a signed distance field instead of a voxmap for smooth and stable contact forces. We are the first to propose this particular contact model for haptic rendering. We claim this model has good approximation properties for haptics. Furthermore, the model converges to a well-defined limit with progressively finer pointshells, and this limit has a very intuitive volume integral interpretation. We show how to evaluate pointshell contact normals under large deformations at haptic rates. We introduce a multi-resolution nested haptic pointshell, and give a deterministic algorithm for graceful degradation of contact; i.e., an upper limit on the computation time of a haptic cycle can be guaranteed. Our algorithm is able to adapt the contact force accuracy to both the difficulty of the current contact configuration (such as the number and sizes of the current contact areas) and the speed of the particular computer. We improve static virtual coupling by handling large rotational differences between the manipulandum and the simulated object. We also suggest static damping to improve simulation stability, and demonstrate how to handle virtual coupling gradient singularities with the saturated virtual coupling model. We give detailed descriptions of several important aspects of a 6-DOF haptic simulation, such as virtual coupling saturation, limiting simulation velocities and handling degenerate virtual coupling gradients. Our description is complete and enables one to build high-quality 6-DOF haptic simulations of (reduced-deformable) objects with complex geometry in contact.

Chapter 2

Real-time geometrically nonlinear deformable models using model reduction

2.1 Background: Nonlinear deformable object modeling

Deformable objects have been studied by the solid mechanics community for decades. This community has researched the problem primarily in an offline, non-interactive setting. *Interactive* simulations of deformable models, however, present certain specific challenges. Fast update rates are required since the amount of computation time to compute the solution at the next time step is limited: it is not possible to spend more computation time with certain selected timesteps, such as for example when updating preconditioners. In interactive computer graphics and haptics, the goal is to develop robust methods running at fast update rates, suitable for real-time computer simulation. The fields of solid mechanics and interactive computer graphics obviously overlap, addressing two facets of the same problem. This thesis makes significant use of solid mechanics to address computer graphics problems. It also presents certain contributions to solid mechanics deformable object modeling. We now give a tutorial on deformable object modeling with FEM.

2.1.1 The finite element method

The partial differential equations (PDEs) of continuum mechanics describe how physical objects deform under the presence of external forces. These equations can model both *static response* and *deformable model dynamics*. Static simulations only give the rest deformation under given external forces, while deformable dynamic simulations include the effects of deformable object mass inertia and damping, and can as such simulate the *transient response*: vibrations resulting from a given time-evolution of external forces. In both cases, the equations are nonlinear, and analytical solutions do not exist except in a few basic cases with simple geometry (sphere, cylinder, etc.). There exist a variety of approaches to convert the continuum PDEs to a discretized system of ordinary differential equations (ODEs), which can then be time-stepped numerically. Common discretization approaches include the Finite Element Method (FEM), Finite Difference

Method and Finite Volume Method. Also, in cases where larger errors in the simulation can be tolerated, approximations such as mass-spring systems have been used to approximate interactive deformable objects [GM97]. FEM is widely considered a reliable and accurate (assuming the mesh is fine enough) method to simulate deformable objects, and its main shortcoming is that it is computationally demanding. It also requires knowledge of solid mechanics, and as such, FEM deformable objects are often considered nontrivial to implement. The methods of this thesis are designed, tested and implemented for FEM, but the reduction ideas can also be applied to other discretizations methods.

In the Finite Element Method, the object to be simulated is represented as a 3D volumetric mesh, by dividing the object into a large number of *elements*. The deformation is prescribed (controlled) by the displacements of the vertices of the mesh. Displacement at an arbitrary position in the 3D solid continuum is then defined via interpolation. Such an approximation will of course only be accurate if elements are sufficiently small that the interpolation is able to sufficiently describe the deformations within each element. There exist several different types of elements (tetrahedral, hexahedral, etc.) with various interpolation schemes (first order, higher order). Also, elements can have higher-order shapes (Bézier patches, splines, etc.). There are specific benefits and disadvantages to choosing a specific element type/interpolation scheme, but this choice is not relevant to this thesis: the model reduction applies regardless of the particular choice of the element type. We use the first-order hexahedral 3D "brick" element (with first-order shape, i.e., faces are flat) in our experiments.

By inserting the interpolation functions (also called *shape functions*) into the partial differential equation of motion, one obtains a system of ODEs, where the unknowns are now the displacements of the vertices of the mesh. This system is in general coupled, nonlinear, sparse, numerically stiff, and high-dimensional for 3D meshes with many vertices. Solution methods for these systems are obviously of great practical importance and have been studied thoroughly in the solid mechanics literature. There exists freeware (CalculiX) and commercial computer software (Abaqus, Nastran) to solve such systems. The solid mechanics literature addresses the problem in an *offline* setting: the solution algorithm is not constrained to produce the values for the next timestep within a fixed computation time window, as is the case with interactive applications. Also, these methods are often designed to run on supercomputers, where network latency would present a challenge for an interactive application. Another difference to interactive simulations is that offline methods often see the whole time-evolution of external forces at the beginning of the simulation (an exception would be an offline contact simulation). In an interactive application, external forces are not known in advance, since they depend on user interaction. In general, the motion of a deformable body depends on several factors:

- the initial conditions (initial position and velocity of every individual material point within the deformable object),
- the time-evolution of applied external forces (if any),
- the geometry of the object (the mesh),

- the material properties of the object: mass density, particular material deformation law. Material properties can vary spatially within the object (heterogeneous material properties).

Another important distinction is whether the deformable model possesses *rigid degrees of freedom*. If such degrees of freedom are present (typically six for 3D simulations), the object can undergo both rigid body motion and deformations simultaneously [Sha05]. If enough vertices of the model are constrained to fixed spatial positions, the model loses its rigid degrees of freedom and becomes a purely deformable model. This case is computationally easier and is commonly referred to as *structural dynamics*. Our reduced deformable method is designed for structural dynamics, but we also provide an extension to nonlinear deformable models undergoing rigid body motion.

Short literature guide

An introduction to continuum mechanics for deformable objects can be found in [Sha90] and [Fun77]. Background on nonlinear solid mechanics can be found in [Bel01, BW97, Hol00b]. Personally, I used [Sha90] and [BW97] the most. I also consulted [Hol00b] when I ran into difficulties with following [BW97]. For a computer graphics person interested in learning about FEM deformable objects, I recommend starting by reading the following papers: [PDA01, OH99, CGC+02a].

2.1.2 Strain and stress

When objects deform, internal elastic forces appear in the material, typically trying to restore the object back to its rest shape. Clearly, these internal forces depend on how the object is deformed. The first assumption (a rather trivial one and often not even explicitly stated) made virtually everywhere in solid mechanics literature is that the internal forces at material location X depend only on the deformations (including their time history) *in some arbitrarily small neighborhood of* X. That is, for forces at X, it does not matter how the material is deforming or previously deformed in the vicinity of some other material point $Y \neq X$. Internal forces are therefore a local phenomenon, which naturally leads to investigation of local quantities such as spatial derivatives of deformation.

One of the important goals of computational solid mechanics is to predict (compute) the internal elastic forces for any given deformation configuration. To build such models, it is necessary to define some measure of the "local amount of deformation", as applied to any infinitesimal piece of solid continuum inside the object. The deformation vector $u = u(X) \in \mathbb{R}^3$, which measures the current 3D displacement of the material point X, is not a good measure as its components can become arbitrarily large under rigid translations, which are free of deformation. A better deformation measure is the gradient of the deformation vector, $F = I + \frac{\partial u}{\partial X}$, which is invariant under rigid translations. It is however, not invariant under rigid rotations. For example, for a pure rotation R, we have $F = R$. One commonly used measure which is invariant under any rigid body motion is *the strain tensor*, defined as $\mathbf{E} = \frac{1}{2}(F^T F - I)$. Both F and $\mathbf{E}$ appear in solid mechanics and computer graphic literature when defining stresses/internal forces. Observe that $\mathbf{E}$ is a nonlinear (quadratic) function of u. This strain tensor is sometimes also called the *Green-Lagrange strain tensor*, to distinguish it from the linearized version of strain (*Cauchy strain tensor*) which

is sometimes used for computational speed. Also note that $\mathbf{E}$ is called a "tensor", even though in practice it is often viewed just as a 3×3 symmetric matrix, expressed with respect to the standard basis of $\mathbb{R}^3$. In this thesis, we use the Green-Lagrange strain tensor to measure the local amount of deformation. There are also approaches where F is used directly [BW97, ITF04].

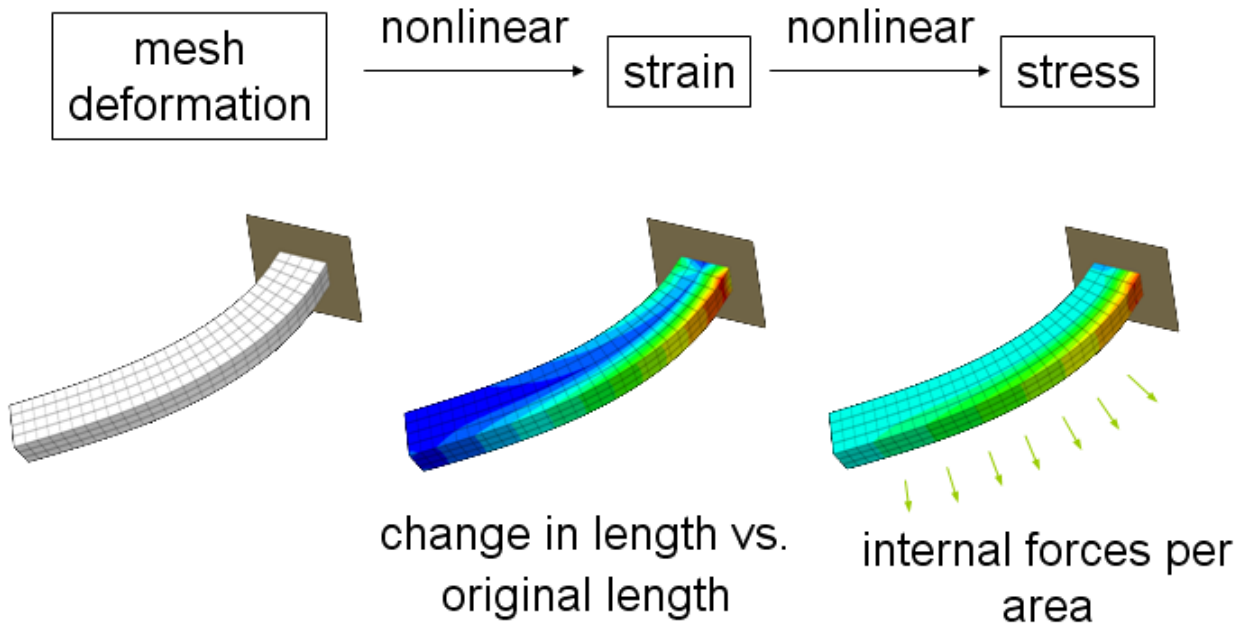


Figure 2.1: **A deflected cantilever beam:** Left: the deformed shape, Middle: strain, Right: stress. Both strain and stress are rank-2 tensors (3x3 matrices for all practical purposes in this thesis); the plotted color gives the maximum eigenvalue of the 3x3 matrix.

Note that both F and $\mathbf{E}$ are defined uniquely based on the local deformation field u in the vicinity of X. They are purely geometric measures, that is, they do not depend on material properties of the mesh such as moduli of elasticity, compressibility, or similar. So, in order to go from geometric deformations to actual forces, we need to choose a particular *material deformation law* (also called the "constitutive equation"). This law will encode physical properties such as material elasticity, compressibility, plasticity, etc. In particular, a material law relates the strain tensor of an infinitesimal piece of solid continuum to its *stress tensor*. Just like strain, the stress tensor can be seen as a 3×3 matrix. It is closely related to the internal forces acting on the material. Imagine a small infinitesimal piece of material of volume dV, and imagine the (imaginary) boundary separating it from the rest of the material. Then, due to deformation, the rest of the material is pulling on this infinitesimal piece, with some net force $f_{\text{internal}}dV$. This internal force equals $f_{\text{internal}} = \text{div}\sigma$, where σ is the stress tensor. Actually, there are three related commonly encountered formulations of stress: the Cauchy stress tensor σ (a very commonly used form of stress), the first Piola-Kirchhoff stress tensor $\mathbf{P}$, and the second Piola-Kirchhoff stress tensor $\mathbf{S}$. The three tensors differ in whether they refer to forces and surface areas in the deformed or undeformed configuration. There are sim-

ple formulas to convert from one to another, see, for example [BW97]. For small deformations, where quadratic terms in the strain tensor can be neglected, all three stresses are equal.

The particular material law chosen defines the behavior of the object under deformation. It can take many different forms, and can in general vary both spatially and temporally, both of which are beyond the scope of this thesis. In the next section, we discuss some common families of material laws.

2.1.3 Elastic and hyperelastic deformable models

In a general material, stresses depend not only on the current value of strain, but also on the entire history of strain. That is, the internal forces depend not only on the current deformation, but also on how the deformable object reached a particular deformed configuration. Also note that as the external forces deform the object, they perform mechanical work on the object. Some of this work is dissipated as heat through material damping, some is converted into deformable kinetic energy and the rest is stored in the object as *internal strain energy*.

Elastic materials are materials for which the stress tensor is memoryless: at any given material point X, the stress tensor is a unique function of the *current* deformation gradient F. The stress-strain law is constant in time for every material position X, but can vary spatially with X (non-homogeneous material laws). It does not matter along what trajectory the material deformed in order to reach the current state, the internal forces will be the same either way. An example of an inelastic material is material that underwent a plastic deformation: if you push the material back toward its original rest shape, it will resist and internal forces will point (roughly) in the opposite direction to the direction during the original deformation; therefore stresses before and after plastic deformation are different, for the same shape (and hence same F). Another inelastic example is human muscle undergoing activation, where the longitudinal stiffness changes with time.

Elastic materials can undergo hysteresis effects: if the deformable body makes a loop in the deformation space, the net strain energy gain will not necessarily be zero. Of course, energy conservation laws must be obeyed. For example, if the net strain energy gain over a loop is negative, this has to manifest as heat, or as increase in deformable object kinetic energy, or as mechanical work performed to the environment. *Hyperelastic materials* are a subset of elastic materials for which also the *elastic strain energy* is memoryless: it is a unique function of body deformation only, and not of the particular deformation path that was followed to reach this deformation. This therefore excludes hysteresis effects. For hyperelastic materials, the internal strain energy equals

$$\int_{\Omega} \mathbf{S} : \mathbf{E} dV, \tag{2.1}$$

where integration is performed over the reference (i.e., undeformed) configuration Ω. Here, $\mathbf{S}$ is the second Piola stress tensor, $\mathbf{E}$ is the Green-Lagrange strain tensor, and colon denotes tensor contraction, i.e., element-wise dot product $\mathbf{S} : \mathbf{E} = \sum_{i=1}^{3} \sum_{j=1}^{3} S_{ij} E_{ij}$. Equation 2.1 follows from the fact that the second Piola stress tensor is work-conjugate to $\dot{E}$ [BW97], and the path-independence of internal strain energy for hyperelastic materials.

A hyperelastic material is *isotropic* if at any position in the body material properties are invariant to local rotations. That is, if the internal forces under a deformation gradient F equal f, they still equal f under a deformation gradient $F \cdot R$, where R is a local material rotation. Anisotropic materials are more "stretchable" in some directions than in others. For example, human muscles are typically stiffer along their axis than in the plane of their cross-section.

2.1.4 Linear materials

In a linear material, the relationship between strain and stress is assumed to be linear. More precisely, the second Piola stress tensor is assumed to be a linear function of the Green-Lagrange strain tensor (note that the first Piola stress tensor and the Cauchy stress tensor are not linear functions of strain, even for linear materials). Since both stress and strain essentially have 6 undetermined components (not 9; they are symmetric matrices), a general linear material can be parameterized as a 6×6 matrix of coefficients relating strain to stress. It can be shown that a linear material is always hyperelastic. Its strain energy function always takes the shape of a fourth-order polynomial in the components of u. This can be seen as follows: the deformation gradient F is always linear in u, therefore the Green-Lagrange strain tensor $\mathbf{E} = (F^T F - I)/2$ is always quadratic in u. For linear materials, the second Piola stress tensor $\mathbf{S}$ is a linear function of $\mathbf{E}$, therefore $\mathbf{S}$ is also quadratic in u. The contraction $\mathbf{S} : \mathbf{E}$ from Equation 2.1 is therefore fourth-order in u, and therefore the internal strain energy is also fourth-order in u.

If one further assumes an *isotropic* linear material, it turns out that only two parameters out of the original 36 remain. This material is called the St.Venant Kirchhoff (StVK) material. It is commonly used in computer graphics for its simplicity. For example, the following papers all use StVK: [DDCB01, CGC+02a, OH99] (they use an alternative name for it, but mathematically, they use the same model (StVK)). StVK is defined by a linear stress-strain relationship as follows:

$$\mathbf{S} = \lambda(\operatorname{tr} \mathbf{E}) I_3 + 2\mu \mathbf{E}, \tag{2.2}$$

where $\mathbf{S}$ is the second Piola-Kirchhoff stress tensor, $\mathbf{E}$ is the Green-Lagrange strain tensor, I_3 is the 3×3 identity matrix, and λ, μ are (possibly spatially varying) Lamé constants. The Lamé constants can be expressed in terms of the perhaps more familiar parameters of Young's modulus $E > 0$ and Poisson ratio $\nu \in (0, 1/2)$ as follows:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}. \tag{2.3}$$

The representations (λ, μ) and (E, ν) are equivalent. Equation 2.3 can be used to convert from (E, ν) to (λ, μ), and it is easy to derive formulas for conversion in the other direction.

The disadvantage of StVK material is that it is not suitable for applications involving large compression [BW97]. For example, if a cube (a single FEM element) of unit length is fixed on one face, with the opposite four vertices free, and a compressive load is (slowly; only interested in static response for this mental experiment) applied to face containing the four free vertices, the

reactive elastic force initially grows (as expected), but peaks when the cube is squashed to about 57% of its initial length. If one continues to compress the cube, the reactive elastic force, while still pointing in the right direction, grows weaker, until it completely vanishes when the cube is completely squashed down to a rectangle. If one then continues increasing the force load, the cube will invert. This is an extreme compression example of course. In many applications in computer graphics volume compression is not large and large deformations under this material law typically look visually plausible. Also, StVK does not suffer from a similar problem under extension; if the cube is stretched, the responsive force grows arbitrarily large (as a cubic polynomial). In summary, StVK is a hyperelastic linear isotropic material: elastic strain energy is a unique function of body deformation only (and not of deformation history), and at any location, material is equally stretchable in all directions.

A general (potentially anisotropic) simulation with linear materials is sometimes referred to as *geometrically nonlinear simulation*. This is a term commonly encountered in solid mechanics literature (and also in computer graphics). Anisotropic linear simulations are useful when a material allows more stretching/compression in one direction than in others. Various human tissues or fiber composites are typical examples.

The methods in this thesis apply to general linear materials, not just to StVK. Our runtime simulations are equally fast regardless of whether the isotropic model is used, or a general linear model. We however, use StVK in all of our examples. With anisotropic models, precomputation would take (a constant factor) longer, since more terms need to be assembled when constructing the coefficients of the the internal force, or tangent stiffness matrix polynomials (or also strain energy).

2.1.5 Nonlinear material models

Numerous nonlinear material laws have been presented in the solid mechanics literature. For example, the neo-Hookean material has been used to model large deformations of plastic and rubber substances. In computer graphics, [TBHF03] used an anisotropic nonlinear material model to simulate human muscle tissue. We do not present real-time simulations of non-linear materials in this thesis. Model reduction does apply to non-linear materials [KLM01], but the reduced internal forces no longer have a simple polynomial form and as such cannot be evaluated at interactive rates for detailed models.

2.1.6 Why the simulation is nonlinear

There are several sources of nonlinearity in the equations of motion of a deformable object:

- Geometric nonlinearity: The Green-Lagrange strain tensor is a *quadratic* (hence nonlinear) function of displacements. If strain is approximated by a linear function, one obtains a purely geometrically linear model, which is inaccurate for large deformations. Simulating

geometric nonlinearities produces plausible large deformations, and as such this was the approach adopted by many large-deformation papers in computer graphics [OH99, DDCB01, CGC+02a], including our paper on reduced StVK simulations [BJ05].

- Material nonlinearity: The material law relating strains to internal forces can be nonlinear. This thesis does not address this case directly.

- External forces are usually a very unpredictable time-signal, and as such a very nonlinear function of time.

- Boundary conditions can be a nonlinear function of time. For example, the user could be prescribing positions of certain vertices interactively at runtime, which is a nonlinear constraint. A more difficult case is that of a deformable object colliding with another (say rigid) object, where the contact handling algorithm needs to apply appropriate non-penetration and non-stick boundary conditions to the deformable body. This is a severe nonlinearity, and approximations such as penalty forces are often used as a computationally simpler substitute. Penalty forces avoid the need for any kind of explicit boundary conditions, since they "convert" boundary conditions into external forces. We use penalty forces in our contact algorithm presented in Chapter 3.

Geometrically linear models (that is, using linear strain) can model only small deformations accurately. They are sufficient for certain engineering applications, especially when the deformations are small compared to the size of the elements (such as with earthquake simulations). Geometrically linear models are also popular because linear equations of solid mechanics are more easily solvable than the more accurate nonlinear equations. Also, certain materials (*brittle materials*) behave elastically only for small deformations anyway (and fracture otherwise). In computer graphics, however, one is often interested in deformations that are large, visible, and exhibit interesting dynamics. Think of tissue deforming during surgery, bridges swaying in the wind, plants deforming in a computer animation. In this case, one cannot neglect the nonlinear terms in the equations of motion without introducing very visible artifacts.

Each of these nonlinearities adds additional complexity to the equations that need to be simulated. Model reduction in this thesis applies to the first three sources of nonlinearities. Prescribing positions of vertices at runtime (nonlinearity in boundary conditions) could be addressed using Lagrange multipliers, in a way similar to [CGC+02a]. Of course, since the number of deformable degrees of freedom is limited, one cannot simultaneously constrain a large number of vertices. I believe one day hardware and algorithms will progress to the point where it will be possible to simulate all four sources of nonlinearity at haptic rates. Such a simulation is, however, beyond reach with current algorithms and hardware.

It is difficult to accelerate the process of simulating nonlinear deformable objects, because the computations are extensive, nonlinear, and appear to permit no shortcuts. This is especially the case when using a single CPU, and when accurate results are sought (modulo numerics). Figure 2.2 shows the result of one such unreduced simulation. The solution is accurate (modulo the

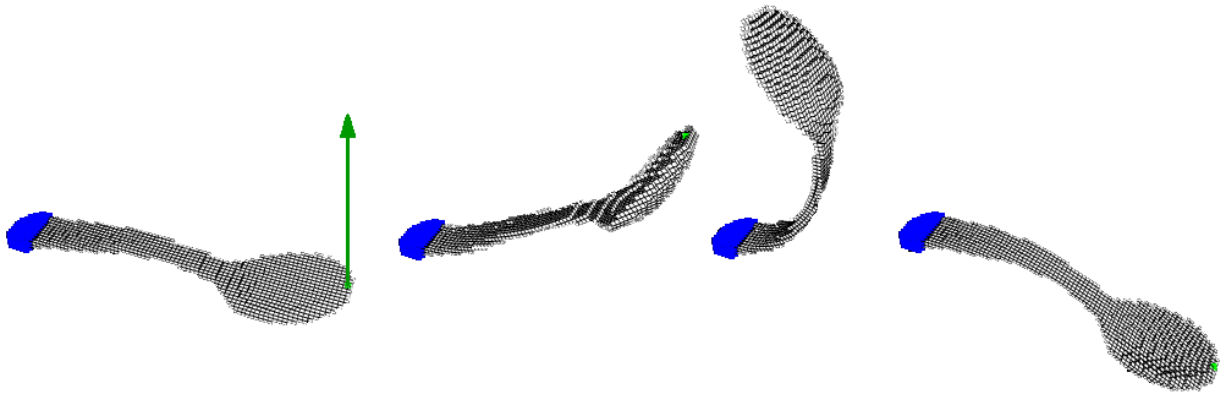


Figure 2.2: **Nonlinear FEM simulation of a voxelized elastic spoon:** The spoon consists of 2005 elements (voxels in this case) and has 11,094 degrees of freedom. Motion was computed using an unreduced approach: direct internal force and stiffness matrix evaluation, combined with the implicit Newmark timestepping scheme and a conjugate gradient solver. Spoon is constrained at the far end (blue), and the motion is induced by a short initial vertical impulse (green). Left-most picture is the rest pose, followed by three representative frames of the dynamic simulation (free vibration).

FEM discretization error, and error due to floating point arithmetic and numerically solving linear systems of equations), but the amount of computation is significant, prohibiting interactive applications for models with more than about one thousand elements. For comparison (see Figure 2.3), a *reduced* simulation of the spoon model under the same initial impulse, with 12 degrees of freedom, and automatic basis selection, achieved a speedup of 50,000x, albeit with some loss of accuracy.

2.1.7 The equations of motion

The *unreduced* equations of motion for structural vibrations of a volumetric 3D deformable object, under the FEM discretization, can be derived from the principle of virtual work of Lagrangian mechanics. These equations of motion (called the *Euler-Lagrange equation*) are a second-order system of ordinary differential equations

$$M\ddot{u} + D(u, \dot{u}) + R(u) = f. \tag{2.4}$$

Here, $u \in \mathbb{R}^{3n}$ is the displacement vector (the unknown), $M \in \mathbb{R}^{3n,3n}$ is the *mass matrix*, $D(u, \dot{u}) \in \mathbb{R}^{3n}$ are damping forces, $R(u) \in \mathbb{R}^{3n}$ are internal deformation forces, and $f \in \mathbb{R}^{3n}$ are any externally applied forces. The mass matrix is constant in time and depends only on the object's mesh and mass density distribution in the rest configuration. In general, it is a sparse non-diagonal matrix, however for algorithmic convenience, it is often simplified into a diagonal matrix by accumulating all the row entries onto the diagonal element (*mass lumping*). Such lumping essentially means that all elements re-assign their volumetrically distributed mass to their vertices: it is as if

the model consisted of a point-like mass at every simulation vertex, with zero mass anywhere else inside the elements. Such a construction of course means losing some simulation accuracy; the accuracy loss is smaller with finer meshes, and is generally not considered a serious problem in computer graphics. Our approach can handle both lumped and non-lumped versions of the mass matrix. Internal forces corresponding to the displacement u are given by the vector $R(u) \in \mathbb{R}^{3n}$. The mapping R is nonlinear due to the nonlinearity of the Green-Lagrange strain tensor, and (in general) due to any material nonlinearities. The mapping R is independent of time for elastic materials, likewise the damping mapping D is independent of time for materials with memoryless damping. For elastic materials, apart from u, the only time-dependent term in the equation is the vector of external forces f, used to model, e.g., user interactions or collision response.

Let $K(u) \in \mathbb{R}^{3n,3n}$ denote the Jacobian matrix (the gradient) of the internal forces R, evaluated at u. This matrix is called the *tangent stiffness matrix*. The tangent stiffness matrix at the origin $K = K(0^{3n})$ is often called simply the *stiffness matrix* (here 0^{3n} denotes the $3n-$dimensional zero vector).

2.1.8 Cubic polynomials for linear materials

For hyperelastic materials, the internal elastic forces $R(u)$ equal the gradient of the strain energy with respect to the components of the deformation vector. This follows from the principles of Lagrange mechanics. As derived in Section 2.1.4 (page 38), the strain energy of a *linear material* (i.e., a geometrically nonlinear simulation) is a fourth-order multi-variate polynomial in the components of the deformation vector u. The terms of this polynomial are sparse: displacements of two vertices can only appear together in a polynomial term if the two vertices are the same vertex or immediate neighbors in the mesh. Consequently, each component of the unreduced force is a third-order multivariate polynomial function in the displacements of the vertex and all its immediate mesh neighbors.

For the StVK material, these polynomials are given as follows [CGC+02b]. Let e be a mesh element and c one of its vertices. Let $u_a \in \mathbb{R}^3$ denote the deformation of a mesh vertex a. Then, the unreduced StVK force on c, under the deformation vector $u \in \mathbb{R}^{3n}$, as per arising from material

within element e, is

$$\begin{aligned} f_c(u) &= (A_1^{ca} + A_2^{ac} + B_1^{ac} I)u_a + && (2.5)\\ &\quad (\frac{1}{2}C_1^{cab} + C_2^{abc})(u_a \cdot u_b) + (u_b \otimes u_a)(C_1^{abc} + C_2^{cab} + C_2^{bac}) + && (2.6)\\ &\quad (\frac{1}{2}D_1^{abcd} + D_2^{acbd})(u_a \cdot u_b)u_d && (2.7)\\ A^{ab} &= \int_e \nabla\phi_a \otimes \nabla\phi_b \, dV \quad \in \mathbb{R}^{3,3} && (2.8)\\ B^{ab} &= \int_e \nabla\phi_a \cdot \nabla\phi_b \, dV \quad \in \mathbb{R} && (2.9)\\ C^{abc} &= \int_e \nabla\phi_a (\nabla\phi_b \cdot \nabla\phi_c) \, dV \quad \in \mathbb{R}^3 && (2.10)\\ D^{abcd} &= \int_e (\nabla\phi_a \cdot \nabla\phi_b)(\nabla\phi_c \cdot \nabla\phi_d) \, dV \quad \in \mathbb{R} && (2.11) \end{aligned}$$

$$A_1^{ab} = \lambda A^{ab}, \quad A_2^{ab} = \mu A^{ab}, \quad B_1^{ab} = \mu B^{ab}, \quad C_1^{abc} = \lambda C^{abc},$$
$$C_2^{abc} = \mu C^{abc}, \quad D_1^{abcd} = \lambda D^{abcd}, \quad D_2^{abcd} = \mu D^{abcd}. \qquad (2.12)$$

Here, $\phi_a = \phi_a(X)$ denotes the FEM shape function corresponding to vertex a, i.e. $\phi_a(a) = 1$ and $\phi_a(b) = 0$ for $a \neq b$. Einstein summation convention was used; summation runs over all vertices, all pairs of vertices, or all triples of vertices of element e. Most vertices in the mesh are vertices in several elements (the 1-neighborhood of a vertex); the total force on such a vertex is obtained by summing the contributions of all the neighboring elements. Lamé constants λ and μ relate to Young's modulus E and Poisson ratio ν as follows:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}. \qquad (2.13)$$

The coefficients A, B, C, D and parameters λ, μ, ρ can in general vary from element to element (i.e. to support non-homogeneous StVK). If implementing the above equations on a computer, be careful if you need to evaluate the above integrals using a substitution, such as, for example, if you need to integrate over a scaled version of an element. Namely, such a substitution will give a cubic scaling factor with the differential dV, and it will *also* introduce scaling factors into the gradients $\nabla\Phi$. Forgetting to appropriately modify the gradients will result in a difficult implementation bug, as the simulation will still somewhat work, but not very well for large deformations.

Also note that the above equations for internal forces can be generalized beyond FEM shape functions to arbitrary deformation fields $\phi = \phi(X)$. All that is required is to write $\lambda = \lambda(X)$ and $\mu = \mu(X)$ (for non-homogeneous materials that can spatially vary even within elements), then carry the scalars λ and μ into the integrals for $A_1, A_2, B_1, C_1, C_2, D_1, D_2$, and integrate over the entire mesh as opposed to just a single element.

General linear materials: In this case, the second Piola-Kirchhoff stress tensor $\mathbf{S}$ is modeled as a linear function of Green-Lagrange strain $\mathbf{E}$:

$$S_{ij} = C_{ijkl}E_{kl}, \tag{2.14}$$

where S_{ij} is the component of $\mathbf{S}$ at location (i, j), for $i, j = 1, 2, 3$. E_{kl} is the component of $\mathbf{E}$ at location (k, l), for $k, l = 1, 2, 3$, and $C_{ijkl} = C_{klij} = C_{jikl} = C_{ijlk}$ is the rank-4 *elasticity tensor*. There are 36 free degrees of freedom in C. We derived the unreduced internal force on c, under the deformation vector $u \in \mathbb{R}^{3n}$, as per arising from material within element e, to be

$$\begin{aligned} f_c(u) &= \Big(\frac{1}{2}(C_{ijkl} + C_{ijlk})P^{ac}_{lj}e_i \otimes e_k\Big)u_a + \frac{1}{2}\Big(C_{ijkl}Q^{abc}_{klj}e_i\Big)(u_a \cdot u_b) + & (2.15)\\ &\quad (u_d \otimes u_a)\Big(\frac{1}{2}(C_{mjkl} + C_{mjlk})Q^{adc}_{lmj}e_k\Big) + \frac{1}{2}C_{mjkl}R^{abcd}_{kljm}(u_a \cdot u_b)u_d & (2.16)\\ P^{ab}_{ij} &= \int_e \nabla\phi_{ai}\nabla\phi_{bj}dV \quad \in \mathbb{R} & (2.17)\\ Q^{abc}_{ijk} &= \int_e \nabla\phi_{ai}\nabla\phi_{bj}\nabla\phi_{ck}dV \quad \in \mathbb{R} & (2.18)\\ R^{abcd}_{ijkl} &= \int_e \nabla\phi_{ai}\nabla\phi_{bj}\nabla\phi_{ck}\nabla\phi_{dl}dV \quad \in \mathbb{R}. & (2.19) \end{aligned}$$

Here, $\nabla\phi_{ai}$ denotes the i-th component of $\nabla\phi_a = \nabla\phi_a(X)$, for $i = 1, 2, 3$. Note that StVK is obtained for $C_{ijkl} = \lambda\delta_{ij}\delta_{kl} + 2\mu\delta_{ik}\delta_{jl}$.

How were these formulas derived: First, we expressed $\mathbf{E} = \mathbf{E}(X)$ (at any location X inside the material) as a quadratic function of mesh vertex deformations. Then, we applied the material law to express $S = S(X)$ as a (still quadratic due to material law linearity) function of mesh vertex deformations. Next, we used the formula $\mathbf{P} = F \cdot \mathbf{S}$ to express the first Piola-Kirchhoff stress tensor as a cubic polynomial in mesh vertex deformation. Finally, we used the Rayleigh-Ritz virtual work formulation from Lagrange mechanics which states that the generalized force corresponding to the shape function $\phi = \phi(X)$ equals

$$f = -\int_\Omega P : \nabla\phi(X)dV + \int_\Omega f_{\text{ext}}(X) \cdot \phi(X)dV + \int_{\partial\Omega} P^T\phi(X)ndS, \tag{2.20}$$

where Ω denotes the 3D object in the rest configuration. If this principle is applied to the FEM shape function ϕ_c of mesh vertex c, the first term of the above Equation 2.20 gives the formula in Equation 2.15. The second term corresponds to external forces, distributed volumetrically across the object. The last term are the surface tractions applied to the object; we assume that the object boundary is stress-free; therefore, this term is zero. Note that boundary conditions such as vertices with fixed locations are still correctly supported, by removing the appropriate rows and columns from the stiffness matrices and internal force vectors. This construction is not limited just to FEM shape functions: functions $\phi = \phi(X)$ can be arbitrary (potentially non-localized) displacement fields. They could be, for example, polynomial global shape functions used in [PW89, MHTG05].

2.1.9 Offline simulation

When solving the Euler-Lagrange equation, the conventional FEM approaches to solid mechanics (such as those without model reduction) typically perform two distinctive computations in a single timestep. In the first part, given the current model deformation, the algorithm evaluates the internal elastic forces $R(u)$ and potentially (depending on the particular choice of the numerical integrator) also tangent stiffness matrix $K(u)$. To construct $R(u)$ and $K(u)$ essentially the same relatively simple nonlinear computation is performed for every element, with deformations of the vertices of this element as input. The inputs to each such "computational kernel" therefore vary from element to element, but the kernel is the same for all the elements. Each element can be processed independently, and the total computation runs in time $O(n)$ for a mesh with n vertices, with the asymptotic constant depending on the particular mesh topology and the type of element used. Despite the seemingly favorable linear running time, the computation of internal forces is time-consuming and easily makes the simulation non-interactive for large models. The second part of a timestep involves solving a large sparse linear system of equations; typically this is only necessary with implicit integration (another case is if non-lumped mass matrix is used for explicit integration). Again, for detailed deformable models, this task is computationally non-trivial. Typically the system is solved with an iterative solver, such as a conjugate gradient solver, where the cost of each iteration is $O(n)$, with many iterations required for large meshes. It is possible to use preconditioning such as the incomplete Cholesky factorization, however the system matrix changes with deformation (since the tangent stiffness matrix changes), and as such the preconditioner needs to be updated often. The timesteps when such a preconditioner update occurs will require a significantly longer amount of computation than the non-update timesteps, which is incompatible with interactivity. In practice, the internal force/tangent stiffness matrix construction times dominate the computation for small to medium size meshes (e.g. 20,000 elements), whereas the system solve becomes a bottleneck for large meshes, such as meshes with 1,000,000 elements. This is to be expected, since the cost of constructing the internal forces grows linearly, whereas the system solve time scales superlinearly. As mentioned, the system solve can be avoided with explicit integration when the mass matrix is lumped; this is sometimes called *explicit finite elements* [OH99]. It requires small timestep to maintain stability. The advantage is that a linear system solve is not required, and that fast high-frequency phenomena are representable due to small timesteps. For example, this method could simulate localized impact with the subsequent shock waves propagating across the deformable model.

A standard deformable object algorithm will perform such a two-stage computation for every timestep. This is the approach used by most commercial/freeware solid mechanics software packages available today. Figure 2.2 shows the result of one such simulation. The solution is very accurate, but the amount of computation is significant, prohibiting interactive applications for models with more than a couple of hundred elements. For comparison (see Figure 2.3), a *reduced* simulation of the spoon model under the same initial impulse, with 12 degrees of freedom, and automatic basis selection, achieved a speedup of 50,000x, with a modest loss of accuracy.

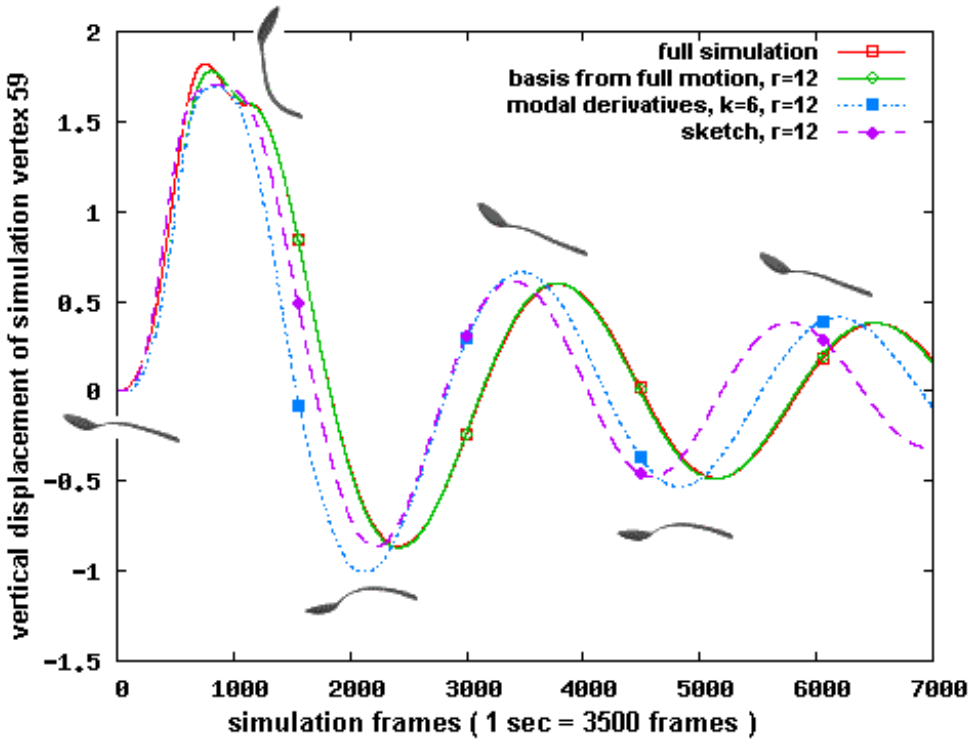


Figure 2.3: **Accuracy experiment:** Vertical displacement of a spoon simulation mesh vertex, located centrally at the tip of the spoon. Some deformations from this experiment were previously shown in Figure 2.2; the plotted simulation mesh vertex is colored green in Figure 2.2. Length of spoon is about 2.5 units. Triangle mesh poses are shown for reference. In all cases, motion is excited by the same identical initial short vertical impulse. The "basis from full motion" curve is obtained by performing Principal Component Analysis (PCA) on the motion computed by the full unreduced $11,094$-dimensional simulation, and then re-simulating the system in this basis. Note that in this case the basis is close to optimal for the simulation, and as such the curve almost aligns with the full simulation curve. This basis, however, wouldn't generalize well to other force loadings. "Modal derivatives" and "sketching" are two novel methods to generate the motion subspace, presented in our paper [BJ05]. The method of modal derivatives is automatic and makes no assumptions on the particular force loadings of the structure. All the reduced spaces are 12-dimensional. Our webpage [JJ] contains a video comparison corresponding to this Figure.

2.1.10 Static simulations

Sometimes, one is interested only in the deformations assumed under a certain fixed static load, as opposed to the dynamics of deformations. Such static simulations are useful, for example, when determining how structures (bridges, buildings) are able to sustain applied loads.

It is easy to convert the Euler-Lagrange equation into a form that supports static simulations: simply discard all terms except $R(u)$ and f. Given an external static load f, the problem then becomes solving the nonlinear equation $R(u) = f$ for u. This is commonly done by the application

of a Newton-Raphson procedure, which is a generalization of the familiar 1D method to solve for the roots of a 1D nonlinear equation. Newton-Raphson involves repeated iterations consisting of evaluating the internal forces $R = R(u)$ and the tangent stiffness matrix $K = K(u)$, followed by a linear system solve $K\Delta u = f - R$ to obtain the next solution approximation $u + \Delta u$.

The model reduction presented in this thesis applies equally well to static simulations as to dynamic simulations. In the case of a static simulation, reduction gives us a *reduced static simulation*. Such simulations can be easily generated using the reduced implicit Newmark integrator (described in Section 2.9): simply discard the reduced mass matrix and damping terms from the reduced equations of motion.

2.2 Related work: Real-time deformable objects

Large deformation models are subject of interest in both interactive computer graphics and offline solid mechanics. Physics-based large-deformation models have been used successfully in graphics for almost two decades [TPBF87, BW92, MT92], and enjoy widespread application in mature graphics areas, such as cloth simulation [BW98, BFA02].

Saint-Venant Kirchhoff (StVK) and (anisotropic) linear materials are often sufficient for the purposes of computer animation. For example, linear materials have been exploited for fast large-deformation kinematics of Cosserat rods [Pai02]. StVK is very common in computer graphics papers: the following papers all use StVK (another term is usually used; e.g., nonlinear simulation or geometric nonlinearities): [ZC00, OH99, PDA01, DDCB01, CGC$^+$02a]. These simulations include geometric nonlinearities and are accurate modulo the FEM discretization, the timestepping discretization and the material linearity assumption. The time complexity, however, is at least $O(n)$ to construct the internal forces (and any required stiffness matrices), since all the mesh elements need to be traversed (n is the number of vertices in the model). As such, simulation costs are proportional to mesh geometric complexity. For example, the runtime assembly of all the cubic force terms for every element limits the interactivity to only about a thousand elements [ZC00, PDA01]. If an implicit solver is used, an additional cost of several $O(n)$ CG iterations will be incurred to solve the large sparse linear system. If the mass matrix is lumped, an explicit solver can be used instead; but this requires small timesteps to maintain stability, and the $O(n)$ cost of computing the internal forces cannot be avoided anyway. In contrast, the reduction approach from this thesis gives runtime costs dependent only on the number of reduced degrees retained, and independent of n.

Local frames of reference can serve as a useful tool to simplify/approximate/accelerate nonlinear simulations. For example, a deformable object undergoing free flight can be approximated by superimposing (small) deformations (timestepped in the local frame of reference) on top of large rigid body motion [TW88, MT92, Sha90]. Closely related to this are so-called "stiffness warping"

methods (c.f. corotational formulations) [MDM+02, MG04, ITF04]. Here, large deformation simulation is achieved by using polar decomposition to extract the rotation that most closely matches the current large deformation of each element. Each element is then (conceptually) "unrotated" to the rest frame of reference, where some amount of deformation will persist due to the non-rigidness of the simulation. Linear forces are calculated with respect to this "deformation on top of rigid body motion", and then rotated forward to the current configuration. The resulting simulation is similar (but different) to a geometrically nonlinear simulation; one difference is that if there is no rotation but only deformation (such as in stretching), stiffness warping gives a purely linear model and will experience volume growth with large deformations. This, however, is not a significant practical problem for many applications: simulations with large rotations plus a small amount of superimposed deformation represent a common class of deformable objects in practice. In a similar manner, polar decomposition can be used to determine particle goal positions in the non-FEM, but very fast models of [MHTG05, RJ07].

All of these local frame of reference approaches, however, require traversing all the elements (or particles) at every timestep to extract the per-element rotations from the deformation gradient (or performing singular value decomposition on the deformation gradient), again incurring at least a cost of $O(n)$ per timestep to construct the internal forces, plus the integration cost.

Multi-resolution methods use hierarchical deformation bases to adaptively refine the analysis based on deformation activity of the model [DDCB01, CGC+02a, GKS02] (see Figure 2.4). The multi-resolution approach is in a sense complementary to the reduced deformation approach. It works by constructing a (spatially) multi-resolution deformation basis, where the basis dynamically grows/shrinks depending on what part of the model currently experiences interesting deformation activity. However, the multi-resolution basis is defined purely based on geometry: it does not use the information on the spatial distribution of material parameters. In contrast, an optimally chosen reduced basis can capture the "typical nonlinear response" of the system, and as such allows the object to deform precisely into the "typical" shapes. Other degrees of freedom need not be included, which can make the basis small and fast. In this sense, a reduced basis is adapted to the equations of motion.

One additional feature/problem with multi-resolution methods comes from the fact that the model is simulated with respect to a coarse mesh everywhere it is not currently refined. At any given moment of time, a significant (large) part of the model will be simulated at a coarse level; if the model were resolved to the finest level everywhere, there would be no benefit of running a multi-resolution simulation, as the simulation would be no different than a standard FEM simulation. Coarser shape functions, however, do not resolve the geometry as fine as the finer shape functions; essentially, they can behave as if all geometry below the current level of detail is filled up with "virtual" material (including gaps of free space). This means that, at any given moment of time during the simulation, the contributions of fine geometric detail (and also any fine-scale material law variation) is lost, for a significant (large) part of the model. In contrast, our method performs the pre-process on the (fine) geometric mesh and the intricacies of fine geometry (and material law distribution) get "baked" into our pre-processed data. While the distinction might not

be as important in several practical cases, consider for example, simulation of fractal geometry, such as a 3D Sierpinski cube: clearly, deformable dynamics will be compromised if parts of the model are filled up with "virtual" material as in a multi-resolution method. Multi-resolution approaches, however, are able to simulate local deformations, and could, in principle, be combined with our reduced deformable models.

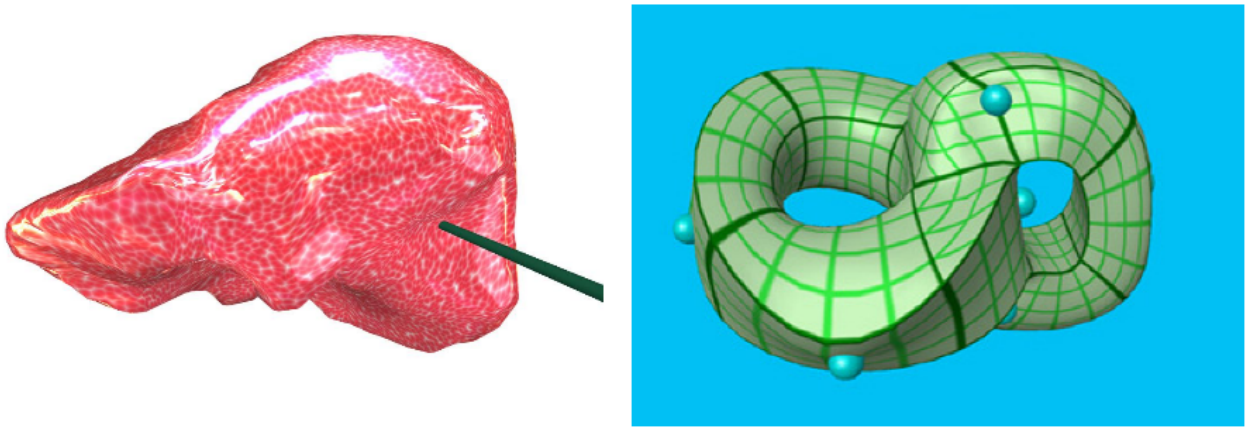


Figure 2.4: **Multi-resolution approaches to simulating deformable objects:** Left: an non-nested multiresolution hierarchy approach of [DDCB01], Right: and a subdivision basis approach of [CGC+02a].

Assumed-shape (low-dimensional) approaches are commonly used in (interactive) computer graphics. For example, high resolution meshes can be deformed using coarse deformable models [PW89, FvdPT97, MG04]. Subspace integration is related to discretizations using global displacement bases that are commonly used in graphics to avoid solving large systems (e.g., during semi-implicit integration), and reducing numerical stiffness (for explicit timestepping), e.g., global polynomial shape functions [WW90, BW92], deformable super-quadrics [MT92], free-form deformation basis functions [FvdPT97]. Multiresolution discretizations also project dynamical equations using multiresolution scaling functions [GKS02].

Linear modal analysis: Every solid object, whether a string, a shell membrane or a volumetric 3D object, possesses natural "eigendeformation" shapes (called *modes*, or linear vibrational modes), with a certain natural frequency for each shape. These shapes of course depend on the object geometry, boundary conditions (such as what parts of the object are fixed), and material properties of the object. For example, the first mode of a guitar string rigidly attached on both ends, and undergoing small transverse deformation, is a half-sine wave; higher modes are sine waves with higher spatial frequencies. The frequency spectrum is in general discrete and countably infinite for true continuum objects, and finite for discretized models, such as those obtained using FEM. The FEM spectrum of course converges to the true spectrum of the continuum object with finer

meshes. In many applications, the lowest natural frequencies and their corresponding shapes give a reasonable description of the deformations of the object. The approximation is convergent in the sense that if more modes are added, progressively better and better accuracy is obtained; if all modes are used, one obtains the same accuracy as under a full model. The technique of computing the first few modal shapes and frequencies of an object, and then using them for (interactive) simulation of deformable objects, is called *linear modal analysis*. In graphics, this technique was first mentioned in [PW89] (but the modes used were polynomial functions up to a given degree, not the actual physical modes), and later popularized by several researchers [OSG02, JP02, HSO03]. Linear modal simulations are fast and easily interactive, even for more than a hundred linear modes. The main drawback is that large deformations look very obviously incorrect: the usage of linear strain causes very visible spurious volume growth under large deformations.

Modal warping was introduced by [CK05a], and it is an approach to extend linear modal analysis to large deformations. As such, it addresses a similar problem to our work. Modal warping starts by precomputing the modes of linear modal analysis (which assumes linear strain and linear materials). At run time, per-element rotations of modal analysis are quickly extrapolated to produce a fast parametric nonlinear shape model. This approach is useful for eliminating gross distortions associated with linear modal analysis. In modal warping, the equations of motion are treated separately from the nonlinear shape: the modes of linear modal analysis are driven by *independent* simple harmonic oscillators, just like in the small deformation case; but a nonlinear shape is displayed and used for collision detection. An initial condition exciting only one of the modes will generate single-mode motion (regardless of amplitude), and hence the well-known nonlinear coupling of modes cannot be captured. In our work, the modes are coupled via an analytic reduction of the StVK finite element model. Our approach uses a reduced displacement basis produced from actual nonlinear shape statistics. Another difference is illustrated by deformations in which no element rotations occur, such as a beam's axial extension mode. Modal warping gives a simulation identical to linear modal analysis in such cases; the volume grows linearly as the beam extends. In our model, forces are cubic polynomials and structure becomes stiffer with extension; modes couple to counteract volume growth. One other benefit of our model is that the displacements are a simple linear function of the reduced coordinates, whereas in modal warping, the relationship is nonlinear, and cannot be evaluated as quickly. Our linear shape model also enables us to accelerate collision detection [JP04], which we use extensively in our haptic simulations (Chapter 3).

Element inversion: StVK models are perhaps the simplest kind of physical large-deformation model, and one well-known deficiency is that forces are inaccurate under larger compressions (see [ITF04] for a discussion). In the worst case, elements may actually invert without proper restoring forces, and suitable steps must be taken to address element inversion. Such an approach has been demonstrated in [ITF04]: the deformation gradient of each element is decomposed using singular value decomposition, and then the stress tensor is computed based on the singular values, such that it gives internal forces restoring the deformations back to the rest shape. We note that an alternative strategy to such element inversion prevention is to remesh the simulation domain when

the aspect ratio of the elements falls below some acceptable threshold; this, however, requires good remeshing strategies, which is a difficult problem on its own. Although the approach of this thesis is not suitable for simulating the general and complex deformations found in Irving et al. [ITF04], it is designed to be substantially faster for interactive applications. Also, the concerns about element inversion are constrained to our precomputation phase, and are not a major concern for runtime subspace integration, since the shape subspace greatly restricts the likelihood of element inversion (but does not preclude it).

Precomputation is used extensively in our method. Appropriate quantities (the motion basis and reduced force polynomial coefficients) are computed in a pre-processing stage, before the actual simulation. The results are cached on disk and then used by the runtime simulation to quickly timestep the solution. To this date, most precomputation-based approaches for real-time simulation have considered geometrically (and materially) linear models. For fast elastostatics, condensation approaches have been used to obtain boundary responses [BNC96], as well as precomputation of boundary Green's function responses [CDA99, JP99]. James and Fatahalian [JF03] precompute nonlinear deformation responses to a finite set of user impulses, and apply dimensional reduction using PCA. Although their approach handles self-collisions, it restricts the range of possible runtime interactions to a small discrete set of pre-selected impulses. On the other hand, our approach allows general runtime forcing within the reduced-dimensional subspace (two force loads with equal subspace projection generate same deformations under our method).

2.3 Related work: Dimensional model reduction in solid mechanics

Dimensional model reduction is a technique to simplify simulation of dynamical systems described by differential equations. Complex systems can be simulated by reducing the dimensionality of the problem, yielding systems of differential equations involving fewer equations and fewer unknown variables. These equations can be solved much more quickly than the original problem, with some accuracy cost to the solution. This method also appears in literature under the names of *Principal Orthogonal Directions Method*, and *Subspace Integration Method*, and it has a long history in the engineering and applied mathematics literature [Lum67]. Static simulations, where dynamic terms in the equation of motion are neglected, were also addressed [NP80, Noo81].

In nonlinear solid mechanics, early methods extended the principle of mode superposition for linear vibration analysis by using local tangent mode superposition [Nic76], and later the derivatives of tangent eigenmode vectors were also included [IC85b]. Explicit computation of the coefficients of reduced force polynomials for a time-varying basis of motion is suggested in [ASB78]. These techniques are not suitable for interactive applications because they periodically involve timesteps with a large amount of computation, such as when the local basis is updated, and the number of derivative modes required for accuracy can grow too quickly to be efficient. Recently, a statistical approach to basis generation for finite element models was presented by Krysl

et al. [KLM01], wherein a full-degree of freedom system is first simulated, and then standard PCA is applied to the resulting deformations to obtain a typical deformation basis. This is a non-interactive technique with external forces known and fixed in advance, and the simulated nonlinear deformations were relatively small compared to deformations in our method. Also, reduced internal forces and reduced stiffness matrices were assembled by first constructing *unreduced* quantities (followed by subspace projection), which is prohibitively expensive for interactive simulation of complex models.

Model reduction beyond solid mechanics: Model reduction has been used extensively in the fields of control theory, electrical circuit simulation, computational electromagnetics and micro-electromechanical systems [LB05]. Applications of model reduction to fluid simulation can be found, for example, in [KKR00, TLP06]. The common theme is to project the original state-space onto a low-dimensional subspace to arrive at a (much) smaller system having properties similar to the original system. Many powerful reduction techniques have been designed, in particular for linear time-invariant systems. Despite this progress, order reduction of large-scale nonlinear systems is still in its infancy [BFSV06].

2.4 Overview of our approach

In the remainder of this chapter we will show how dimensional model reduction on large geometrically nonlinear deformation models with *linear materials*, as commonly used in graphics (for example, the St. Venant-Kirchhoff model, or StVK), can lead to extremely fast and precomputable approximations for real-time applications.

In Section 2.5, we discuss how to generate 3D simulation meshes for "polygonal soup" models of computer graphics. These meshes and the (potentially spatially varying) material parameters form the input to our FEM model reduction algorithm. Assuming a motion basis is known, the Euler-Lagrange equations of motion can be projected on the subspace, as demonstrated in Section 2.6. Dimensional model reduction for linear materials (such as StVK) results in internal force models that are simply cubic polynomials in reduced coordinates. Coefficients of these reduced force polynomials can be precomputed for efficient runtime evaluation of exact internal forces and stiffness matrices (Section 2.1.8).

A key challenge is to construct a good reduced deformation basis for describing general large deformation problems; we discuss this in Section 2.8. To this end, we present two approaches to good quality basis motion generation: modal derivatives and a sketch interface. Modal derivatives provide an automatic approach (requiring no prior simulation data) where the standard linear modal analysis basis is augmented by the derivatives of the linear modal basis vectors. In the sketch-based interface, the user is presented a linear modal analysis model and interacts with it. The imposed forces are recorded, and then an offline FEM solver generates the deformation samples. Both the modal derivatives approach and the sketch approach produce a certain number of deformation fields: the modes and their derivatives for the modal derivatives approach, and the

deformation samples for the sketch approach. We process these deformation fields with a statistical model reduction technique, giving us the nonlinear modal shape basis functions (the empirical eigenvectors). The obvious candidate for model reduction is Principal Component Analysis (PCA); however this would overemphasize deformations in mesh regions where the mesh is very dense (for meshes that do not have regular or near-regular distributions of vertices). Instead, we introduce *mass PCA*, a variant of the PCA data-reduction method which appropriately decomposes the mass matrix and as such effectively correctly weights the different simulation mesh vertices when performing PCA.

The subspace integration costs are independent of geometric complexity. Consequently, large deformation physics can be integrated at extremely fast rates using trusted subspace integrators, e.g., implicit Newmark, while graphical rendering is done at slower rates. We describe the implicit Newmark integrator in Section 2.9. For example, the large bridge example shown in Figure 1.2 can only be dynamically rendered at about 40 Hz, but its dynamics can be integrated at more than a kilohertz, thus enabling haptic simulations of complex large-deformation models. In general, the integration speed is proportional to the number of subspace dimensions employed, e.g., with 4 dimensions the bridge dynamics can be integrated at over 200 kHz. We demonstrate our method on a variety of examples and compare the method accuracy under progressively larger basis sizes (Section 2.13).

2.5 Mesh generation

The deformable body is represented as a volumetric mesh consisting of 3D polyhedra called *elements*. A particular body deformation is specified by the displacements of the volumetric mesh vertices. For a volumetric mesh consisting of n vertices, the *displacement vector* $u \in \mathbb{R}^{3n}$ contains the x, y, z world-coordinate displacements of model vertices. If the deformation vector is known, it is possible to render the deformable object, and perform collision detection. Typically, a small set of vertices are constrained to have zero displacements, which removes the rigid body degrees of freedom (this is the so-called "structural dynamics"). However, we also support unconstrained meshes, through our extension of modal derivatives to free-flying objects (Section 2.8.3).

2.5.1 Choice of elements

We use the first-order hexahedral "brick" element in all of our experiments. This element is a polyhedron with 6 faces (a hexahedron), and the shape functions correspond to trilinear interpolations of the data from the element vertices into the interior of the element. In particular, our bricks are uniform size voxels as described in Section 2.5.2.

Of course, the Finite Element Method provides several other different types of elements. Probably the most common element type used in computer graphics is the first-order tetrahedron element. This is the simplest kind of a FEM element, with a constant value of strain inside each tetrahedron, and engineering references often cite that very fine tetrahedron meshes are necessary

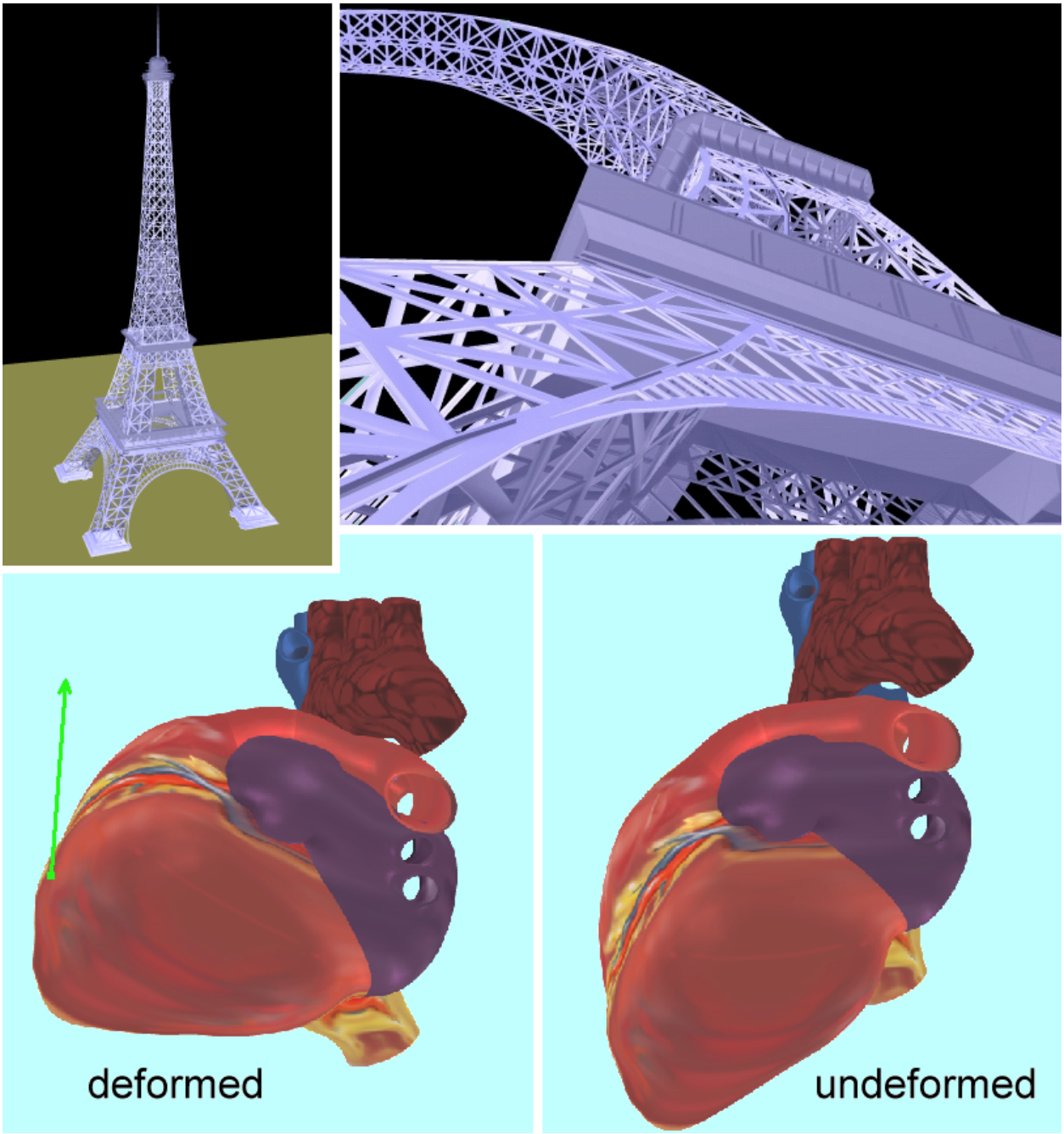


Figure 2.5: **Subspace integration of Eiffel tower and heart models**

to achieve sufficient solution accuracy. Elements also differ in the interpolation order of the shape functions. A standard way to improve the accuracy of the first-order tetrahedron is to add one extra degree of freedom at the midpoint of every tetrahedron edge, resulting in a second-order

tetrahedral element. There are specific benefits/losses/trade-offs to choosing a specific element type/interpolation scheme. In general, choosing a higher-order scheme results in more computation to determine the internal forces/tangent stiffness matrix, however the convergence is faster. An engineering rule of thumb is that computation costs grow linearly with the order of shape functions, but convergence grows exponentially. However, this convergence is impeded in cases where the volumetric mesh is undergoing contact because in such cases, there is a lot of non-smoothness (derivative discontinuities, etc.) in the solution. In such cases, the power of higher-order elements to represent smooth deformation fields is less of an asset. For this reason (and simplicity of programming), first-order elements are still very common in graphics. Another design parameters of a FEM element is the order of its geometric shape; this thesis only uses first-order shape elements (faces are flat), however higher shape order elements, such as those where the faces are cubic Bézier patches would also be possible.

In any case, the element choice is not relevant to this thesis: the model reduction applies regardless of the particular choice of the element type. That is, our basis selection techniques take the initial degrees of freedom of the system, and appropriately reduce them to a low number of reduced degrees of freedom. The particular element shape of course plays the appropriate role in this reduction, as it affects how stresses depend on the original degrees of freedom. However, the process itself is the same for all element types.

2.5.2 Free-form deformations

In computer graphics, it is often useful to simulate models which are essentially "polygon soups," i.e., unorganized, potentially non-manifold surface triangle meshes. However, a good quality volumetric mesh is necessary for simulation. Of course, sometimes the volumetric mesh is already given, in which case we can simply use it for simulation; one can then render the surface of the mesh for visualization. If we are, however, given an input "polygon soup" triangle mesh of an object that we wish to animate, how can we create the corresponding volumetric mesh for simulation?

We follow a common approach in graphics, wherein a 3D volumetric simulation mesh drives the deformations of an embedded triangle mesh. In our implementation, the volumetric mesh is obtained by voxelizing the triangle mesh into tiny elastic cubes (8-node first order brick elements) [JBT04, MTG04]. This is performed by fitting a bounding box to the polygon soup geometry (all sides equal; centered on the model, and enlarged by some factor, e.g. 1.2x), then uniformly dividing the cube (in all three dimensions) into a regular grid of voxels, and storing the voxels that intersect the geometry. While this gives volumetric meshes with a lot of 90 degree angles ("staircase meshes"), these meshes performed well for our purposes. If necessary, inhomogeneous material parameters can be assigned to the cubes, for example to make the cables on a bridge stiffer or leaves on a tree softer than the branches. We do this manually, by visualizing the voxel mesh, selecting appropriate regions, and generating appropriate material files.

Of course, only the original triangle input mesh (call it the *rendering mesh*) is actually displayed on the screen during the simulation. The volumetric mesh is only there for internal calculations.

External forces applied to the system typically come from user interaction, such as the user pulling on a vertex, or they are contact forces from a collision detection/contact algorithm. These forces are usually given with respect to the rendering mesh, not the simulation mesh. It is therefore necessary to transfer these forces appropriately to the simulation mesh. This can be achieved as follows: a force acting on a rendering mesh vertex is distributed barycentrically to the eight simulation nodes of the voxel containing this vertex. In a similar manner, displacements of the volumetric mesh are transfered back to the triangle mesh, so that the deformed triangle mesh can be rendered on the screen. We found such free-form discretization convenient during precomputation, but of course other kinds of free-form deformation schemes could be used too, such as embedding the rendering mesh into an appropriately defined tetrahedral mesh [MBTF03].

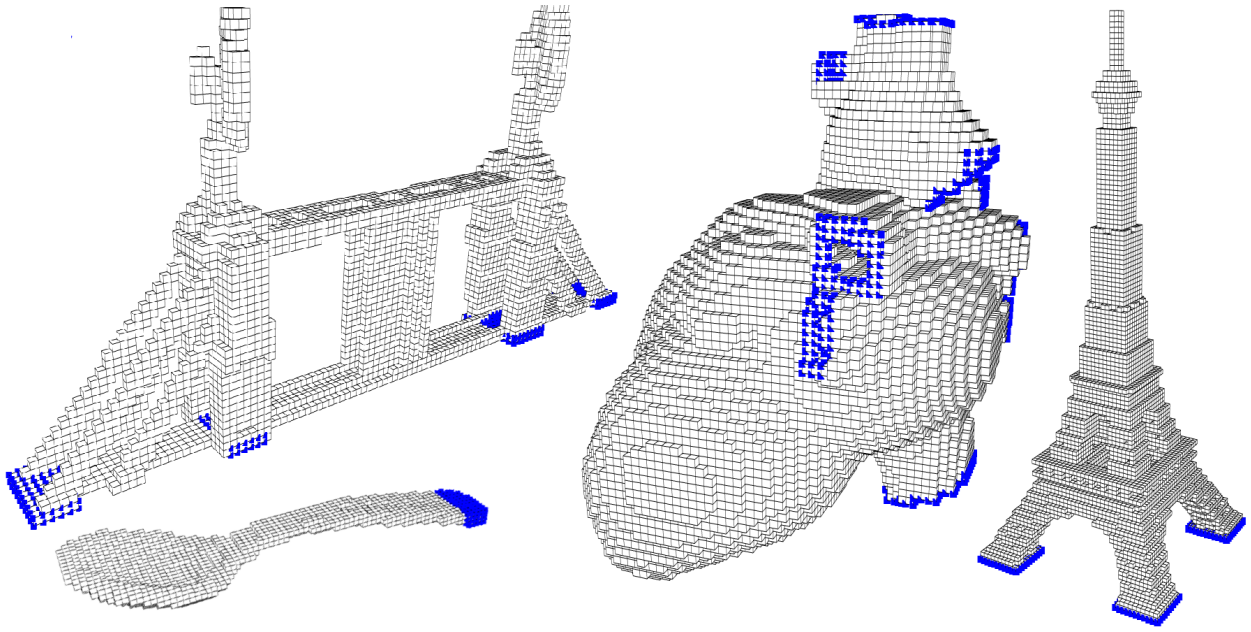


Figure 2.6: **Simulation Meshes:** Blue vertices are constrained.

	rendering		voxel	simulation	
	vertices	triangles	resolution	vertices	elements
spoon	3321	6638	100	3698	2005
bridge	41361	59630	128	11829	5854
tower	45882	105788	140	20713	11304
heart	12186	23616	80	28041	14444

Figure 2.7: **Geometric parameters of models from Figure 2.6.**

2.6 The reduced equations of motion

In a reduced model, the displacement vector is expressed as $u = Uq$, where $U \in \mathbb{R}^{3n,r}$ is some *displacement basis matrix*, and $q \in \mathbb{R}^r$ is the vector of *reduced coordinates*. Here, U is a time-independent matrix specifying a basis of some r-dimensional ($r \ll 3n$) linear subspace of $\mathbb{R}^{3n}$. There is an infinite number of possible choices for this linear subspace and for its basis. Good subspaces are low-dimensional spaces which well-approximate the space of typical nonlinear deformations. The choice of subspace depends on geometry, boundary conditions and material properties. Selection of a good subspace is a non-trivial problem and we will discuss it in Section 2.8. For now, simply assume that a good subspace is available, and that it is specified explicitly by some deformation basis matrix U.

For a given $r-$dimensional subspace of the full deformation space $\mathbb{R}^{3n}$, there are many choices for a specific basis for this subspace, and this choice can impact numerical stability. One choice would be to pick an orthogonal basis, however, it is more natural to make columns of U mass-orthonormal (see Section 2.8.1), i.e., impose $U^T M U = I_r$, where I_r is the $r \times r$ identity matrix. Note that this still does not fully specify U because the columns of U can be rotated with an arbitrary $r \times r$ rotation matrix: if U is mass-orthonormal, so is UQ, where Q is a $r-$dim rotation. We do not select any particular rotations in this thesis. One option would be to pick Q such that the reduced tangent stiffness matrix diagonalizes in the rest configuration; this, however, would not impact execution speed or simulation accuracy.

By inserting $u = Uq$ into Equation 2.4, and premultiplying by U^T, one obtains the *reduced equations of motion*. These equations determine the dynamics of the reduced coordinates $q = q(t) \in \mathbb{R}^r$, and therefore also the dynamics of $u(t) = Uq(t)$:

$$\ddot{q} + \tilde{D}(q, \dot{q}) + \tilde{R}(q) = \tilde{f} \tag{2.21}$$

where $\tilde{D}$, $\tilde{R}$ and $\tilde{f}$ are r-dimensional reduced forces,

$$\begin{aligned} \tilde{D} &= U^T D(Uq, U\dot{q}), && (2.22)\\ \tilde{R}(q) &= U^T R(Uq), && (2.23)\\ \tilde{f} &= U^T f. && (2.24) \end{aligned}$$

Similarly, one can form the *reduced tangent stiffness matrix*,

$$\tilde{K}(q) = U^T K(Uq) U \in \mathbb{R}^{r,r}. \tag{2.25}$$

The existence theorem for systems of ordinary differential equations (ODEs) [HW04a] assures that the system of ODEs in Equation 2.21 has a well-defined unique solution, given a specific instance of initial conditions and time-dependent external forces. Since $r \ll 3n$, the integration of (2.21) is much faster than the integration of the unreduced system (2.4), albeit with some accuracy loss.

2.6.1 Reduced internal forces are cubic polynomials

In order to integrate Equation 2.21 quickly, it is necessary to evaluate $\tilde{R}(q)$ and $\tilde{K}(q)$ quickly, for arbitrary runtime configurations q. For general hyperelastic materials, $\tilde{R}(q)$ and $\tilde{K}(q)$ can be evaluated directly by following Equations 2.23 and 2.25, as was done in [KLM01]. Assuming that unreduced internal forces and the tangent stiffness matrix can be evaluated in time $\Theta(n)$ (which is true as long as the stress-strain relationship can be evaluated in constant time; true for all hyperelastic material models that the author is aware of), the cost to evaluate $\tilde{R}(q)$ and $\tilde{K}(q)$ becomes $\Theta(nr)$ and $\Theta(nr^2)$, respectively.

This cost still has a dependency on n (number of vertices in the simulation mesh). For *linear materials*, we make the cost dependent on r only, by observing that internal forces are in this case cubic polynomials in reduced coordinates. This follows from Equation 2.23 and the fact that $R(u)$ is a cubic polynomial in components of u, as discussed in Section 2.1.8 (page 42). Each of the r components of $\tilde{R}(q)$ is a third-order multivariate polynomial function in the reduced coordinates q, and can be expressed as

$$\begin{aligned} \tilde{R}(q) &= U^T R(Uq) = & (2.26)\\ &= \sum_i P^i q_i + \sum_{j \geq i} Q^{ij} q_i q_j + \sum_{k \geq j \geq i} S^{ijk} q_i q_j q_k, & (2.27) \end{aligned}$$

where $P^i, Q^{ij}, S^{ijk} \in \mathbb{R}^r$ are some constant vector coefficients. If the motion basis vectors (columns of U) have global support (i.e., are non-zero everywhere on the mesh), all polynomial coefficients P^i, Q^{ij}, S^{ijk} are (in general) non-zero, i.e., polynomials are dense. All the models in this thesis have this property (sparse bases would be a natural area of further exploration). Coefficients P^i, Q^{ij}, S^{ijk} can be precomputed, for efficient runtime evaluation. This allows for a fast runtime evaluation of reduced internal elastic forces and the reduced tangent stiffness matrix. We can then simulate the system with a standard *reduced* implicit Newmark integrator, which results in fast simulations of nonlinear dynamics. Simulation costs only depend on the dimensionality of the shape vector space and are as such independent of geometric complexity.

In particular, the coefficients of cubic polynomials of Equation 2.26 can be precomputed as follows. Let e be a mesh element and c one of its vertices. Let $u_a^i \in \mathbb{R}^3$ denote the deformation of vertex a under deformation mode i, for $i = 1, \ldots, r$. Denote the contribution of element e to the global reduced internal force polynomial coefficients by $P_e^i, Q_e^{ij}, S_e^{ijk} \in \mathbb{R}^r$ These contributions can be obtained by inserting the formulas for StVK unreduced internal forces (Equation 2.5, page 43) into Equation 2.23 (Einstein summation convention is used; summation is over all vertices, pairs of vertices, or triples of vertices of e):

$$\begin{aligned} P_e^i &= U_c^T \Big(A_1^{ca} u_a^i + A_2^{ac} u_a^i + B_1^{ac} u_a^i \Big) & (2.28)\\ Q_e^{ij} &= U_c^T \Big((\frac{1}{2} C_1^{cab} + C_2^{abc})(u_a^i \cdot u_b^j) + (u_b^i \otimes u_a^j)(C_1^{abc} + C_2^{cab} + C_2^{bac}) \Big) & (2.29)\\ S_e^{ijk} &= U_c^T \Big((\frac{1}{2} D_1^{abcd} + D_2^{acbd})(u_a^i \cdot u_b^j) u_d^k \Big). & (2.30) \end{aligned}$$

To obtain the global coefficients P^i, Q^{ij}, S^{ijk}, sum the contributions of all the elements:

$$P^i = \sum_{\substack{e \text{ over all} \\ \text{elements}}} P^i_e, \qquad Q^{ij} = \sum_{\substack{e \text{ over all} \\ \text{elements}}} Q^{ij}_e, \qquad S^{ijk} = \sum_{\substack{e \text{ over all} \\ \text{elements}}} Q^{ijk}_e. \tag{2.31}$$

2.6.2 Reduced tangent stiffness matrix entries are quadratic polynomials

The reduced tangent stiffness matrix $\tilde{K}(q) \in \mathbb{R}^{r,r}$ is just the gradient of the internal force vector $\tilde{R}(q)$, and therefore, each component of $\tilde{K}(q)$ is a multivariate quadratic polynomial in q. Specifically, column ℓ of $\tilde{K}$ equals

$$\frac{\partial \tilde{R}(q)}{\partial q_\ell} = P^\ell + (Q^{\ell i} + Q^{i\ell})q_i + (S^{\ell ij} + S^{i\ell j} + S^{ij\ell})q_i q_j. \tag{2.32}$$

2.6.3 Precomputing polynomial coefficients

The coefficients of the cubic and quadratic polynomials from the previous subsections 2.6.1, 2.6.2 can be precomputed as follows. Note that there is one cubic polynomial per reduced force dimension (r cubic polynomials total), and one quadratic polynomial per entry of the reduced tangent stiffness matrix ($r(r+1)/2$ quadratic polynomials total due to symmetry of the stiffness matrix). Precomputation proceeds by first computing all the coefficients of the reduced force polynomials. This can be done in $O(n \cdot r^4)$ time, as indicated by Equations 2.28, 2.29, 2.30, 2.31. It is possible to compute the contributions of each element independently, which makes it possible to compute the polynomial coefficients in parallel on several processors. We routinely used a cluster of 15 machines, for a 15x speedup of the polynomial coefficient precomputation. This was especially useful with larger values of r, e.g., $r \geq 15$. Once the reduced force coefficients have been precomputed, the coefficients of the reduced tangent stiffness matrix polynomials can be obtained easily (Equation 2.32).

Model	r	num elements	precomputation time (single CPU)	size of coefficients
spoon	12	2005	60 sec	98 Kb
bridge	15	5854	186 sec	223 Kb
tower	30	11304	79.2 min	3.0 Mb
heart	30	14444	97.4 min	3.0 Mb

Figure 2.8: **Precomputing polynomial coefficients:** Reported numbers are totals for both reduced force and reduced stiffness matrix, and refer to single-processor computation.

2.6.4 Runtime polynomial evaluation

At runtime, we obtain the *exact* reduced internal forces by evaluating the precomputed internal force cubic polynomials in the current reduced coordinates q. Likewise, we obtain the *exact* cur-

rent reduced tangent stiffness matrix by evaluating the precomputed tangent stiffness quadratic polynomials in the current reduced coordinates q. Since the polynomials are of low degree, dense, and involve many variables (=r), we found no gain in using schemes such as multivariate Horner's polynomial evaluation scheme. Instead, we evaluate the polynomials directly, by assembling $q_i q_j$, and summing the terms together. Even though polynomials involve all possible terms, evaluation order does matter for speed (a speed gain of constant factor), as described next. During pre-process, we organize all the precomputed coefficients of the quadratic terms of the reduced stiffness matrix $\tilde{K}(q)$ into a constant matrix $\overline{S} \in \mathbb{R}^{r(r+1)/2 \times r}$. Each row of this matrix corresponds to one entry of $\tilde{K}(q)$: it contains all the quadratic coefficients of the entry. Note that we are exploiting the symmetry of the reduced tangent stiffness matrix here: the number of rows of $\overline{S}$ is $r(r+1)/2$ and not r^2. Then, to evaluate the quadratic terms of $\tilde{K}(q)$ at runtime, we first assemble $q_i q_j$ for all $i \leq j$ into a vector $\overline{q}$, and multiply $\overline{S}$ by $\overline{q}$. This dense matrix-vector multiplication can be performed efficiently using Level-2 BLAS matrix multiplication routines. A similar scheme was used to quickly evaluate the cubic terms of $\tilde{R}(q)$.

The number of non-highest-order polynomial terms is smaller and their evaluation is faster. We used lower-order schemes analogous to those from the previous paragraph to evaluate these terms. Alternatively, all terms (both highest order and lower orders) could be evaluated in one formula, by augmenting the reduced coordinates vector q to $(q, 1)$ (homogenization). Evaluation of each component of $\tilde{R}(q)$ involves $\Theta(r^3)$ operations, and evaluation of each component of the reduced tangent stiffness matrix involves $\Theta(r^2)$ operations, so both evaluations can be performed in $\Theta(r^4)$ time. Note that evaluation time is independent of the number of vertices and elements in the model. When we discuss the implicit Newmark integrator in Section 2.9, we shall see that one integration step takes $\Theta(r^3)$ time (due to a linear system dense solve with a $r \times r$ matrix), so the time to assemble $\tilde{R}(q)$ and $\tilde{K}(q)$ dominates the timestep, especially for larger values of r.

Model	r	evaluate $[\mu s]$		solve linear	integration	N	time for	graphics frame rate	
		force	stiffness matrix	system $[\mu s]$	total $[\mu s]$		$u = Uq$ $[\mu s]$	standard impl.	GPU-accelerated
spoon	12	8.2	9.5	12.5	30.2	25	565	275 Hz	470 Hz
bridge	15	22.0	25.0	18.4	65.4	10	14500	38 Hz	84 Hz
tower	30	550	770	75	1395	15	25500	17 Hz	40 Hz
heart	30	550	770	75	1395	15	6500	31 Hz	45 Hz

Figure 2.9: **Runtime Computation Performance:** Integration times refer to one integration step. The number of integration steps per graphics frame is N. Timings for the heart and tower models are almost identical due to equal r. Machine used: Intel Pentium 3.0 Ghz processor (manufactured in 2004) with 2Gb RAM.

2.7 Damping

In this thesis, we use a local Rayleigh damping model of the form

$$D(u, \dot{u}) = \Big(\alpha M + \beta K(u)\Big)\dot{u}. \tag{2.33}$$

This damping model is controlled by two positive real-valued parameters, α and β, which, roughly, have the effect of damping low and high time-frequency components of deformations, respectively. This damping model is a generalization of the more familiar linear Rayleigh damping model, which would be obtained if $K(u)$ were replaced by K.

Strain-rate damping (see [DDCB01]) could be used instead of local Rayleigh damping. Such a damping model allows damping material properties to vary arbitrarily across the model, irrespective of the elastic material properties or distribution of mass density. In contrast, the local Rayleigh damping model depends on K and M, so damping cannot behave independently of the elastic properties. Under the *linear* strain rate damping model, the damping second Piola-Kirchhoff stress is a linear function of the strain rate (the time-derivative of the Green-Lagrange strain tensor):

$$S_D = \phi(\mathrm{tr}\dot{\mathbf{E}})I_3 + 2\psi\dot{\mathbf{E}}, \tag{2.34}$$

where quantities ϕ and ψ are (potentially spatially varying) strain-rate constants, analogous to λ and μ. We derived the following formula for the damping force on a mesh vertex c, analogous to Equation 2.5:

$$\begin{aligned} f_c(u) \;=\; & (\overline{A_1}^{ca} + \overline{A_2}^{ac} + \overline{B_1}^{ac} I)\dot{u_a} + & (2.35)\\ & (\frac{1}{2}\overline{C_1}^{cab} + \overline{C_2}^{abc})(\dot{u_a}\cdot u_b + u_a \cdot \dot{u_b}) + (u_b \otimes \dot{u_a})(\overline{C_1}^{abc} + \overline{C_2}^{cab} + \overline{C_2}^{bac}) + & (2.36)\\ & (\frac{1}{2}\overline{D_1}^{abcd} + \overline{D_2}^{acbd})(\dot{u_a}\cdot u_b + u_a \cdot \dot{u_b})u_d, & (2.37) \end{aligned}$$

$$\begin{aligned} &\overline{A_1}^{ab} = \phi A^{ab}, \quad \overline{A_2}^{ab} = \psi A^{ab}, \quad \overline{B_1}^{ab} = \psi B^{ab}, \quad \overline{C_1}^{abc} = \phi C^{abc}, \\ &\overline{C_2}^{abc} = \psi C^{abc}, \quad \overline{D_1}^{abcd} = \phi D^{abcd}, \quad \overline{D_2}^{abcd} = \psi D^{abcd}. \end{aligned} \tag{2.38}$$

Here, quantities $A^{ab}, B^{ab}, C^{abc}, D^{abcd}$ are identically the same as in Equations 2.8- 2.11 (page 43). Reduced strain-rate damping forces are therefore polynomials in q and $\dot{q}$. The polynomials are linear in $\dot{q}$, and the total degree of each polynomial (maximum sum of degrees of q and $\dot{q}$) is again three. The coefficients could be precomputed, again with $O(r^4)$ storage and evaluation time. However, we found local Rayleigh damping model sufficient for our applications.

2.8 Low-dimensional basis selection

The problem addressed in this section is how to select the vector space of shapes that best captures typical nonlinear deformations. But, what are *typical* nonlinear deformations? In the dimensional model reduction solid mechanics approach of [KLM01] they approach the problem statistically: typical nonlinear deformations are taken to be the statistical principal eigenvectors of some pre-existing simulation data. In particular, they first apply some realistic (typically short-lasting) force

to some vertex or set of vertices, and perform (slow offline) unreduced simulation. Next, they process the resulting unreduced deformations using Principal Component Analysis (PCA) to extract a low-dimensional motion basis U, i.e., the "empirical eigenvectors." This allows them to quickly simulate new deformations, within the subspace spanned by U, under force loads similar to those used to obtain U. Such resimulation with loads similar to some previously used loads is known as *re-analysis* in the mechanics literature. Of course, the motion basis is only good for deformations similar to the ones that that were used to generate the basis. One problem with this approach is that for interactive applications, it is unclear what example motion would best describe the essential deformation behavior of future uses. The approach is not automatic, since we cannot simply press a button and build a general purpose model. In this thesis, we suggest two approaches for basis selection: modal derivatives (which is automatic in the sense that it requires no selection of forces and no pre-simulation), and an interactive sketching technique.

The sketching technique is similar to Krysl's re-analysis, with the addition that we make the force selection process more user-friendly by providing immediate deformation feedback from a fast *geometrically* linear model. Modal derivatives have been demonstrated as a natural correction to linear modal analysis vectors [IC85b]. We therefore construct a reduced motion basis from linear modal analysis vectors and their derivatives (appropriately scaled and processed by PCA). Since the linear modal analysis basis forms a natural basis for global small vibrations, the combined linear modes and modal derivatives form a good basis for natural global large deformations.

In both cases, we suggest mass-scaled principal component analysis (mass-PCA) for dimensionality reduction. PCA is not strictly necessary with the modal derivative basis, as the linear modes and their derivatives could simply be concatenated together into a motion basis (followed by mass-orthogonalization, for example using mass-Gramm-Schmidt). However, the total number of derivatives grows quadratic in the number of linear modes, and as such a basis of linear modes and *all* of their derivatives easily exceeds practical basis sizes. We exploit statistical redundancy between the various modal derivatives and linear modes, by applying mass-PCA onto the set of the first k linear modes and all of their $k(k+1)/2$ derivatives. For example, we would often reduce the first 10 linear modes and their 55 derivatives to a motion basis of, say, 15 basis vectors. Both basis generation techniques apply to general nonlinear materials and are not limited to linear materials.

2.8.1 Mass PCA

The goal of this chapter is to define a modification of PCA that fits naturally with our domain data, namely deformations of volumetric meshes. It is possible to simply apply standard PCA, however, for volumetric meshes with varying vertex densities, this will introduce bias into PCA: the deformations in regions with high vertex density will have a higher "vote" in PCA than the deformations in the rest of the mesh. Furthermore, the "proper" metric in the deformation space is not the standard Euclidean metric, but rather the mass-weighted metric; this is the metric that makes standard linear modes orthogonal. Namely, these modes are the eigenvectors of the generalized eigenvalue problem $K\psi = \lambda M\psi$. As is well-known in algebra, eigenvectors of a standard eigenvalue problem $Ax = \lambda x$ are orthogonal, whereas eigenvectors of a generalized eigenproblem $K\psi = \lambda M\psi$ are

M-orthogonal (M is the mass matrix),i.e., the linear modes satisfy $\psi^{jT} M \psi^i = \delta_{ij}$. In general, they are not orthogonal: $\psi^{jT} \psi^i \neq \delta_{ij}$.

Therefore, we need to modify PCA to operate with respect to the mass-weighted inner product, as opposed to the standard Euclidean inner product. First, let us define a mass-based similarity *metric* between two deformation vectors u and v. The standard Euclidean metric is $\|u - v\|_2 = \sqrt{(u-v)^T(u-v)}$, which results in the standard PCA algorithm. Note that this metric essentially originates from the standard Euclidean inner product $(x, y) \mapsto x^T y$. It is not difficult to show that

$$(x, y) \mapsto y^T M x \tag{2.39}$$

defines a mathematically valid inner product on $\mathbb{R}^{3n}$. The resulting metric

$$\|u - v\|_M := \sqrt{(u-v)^T M (u-v)} \tag{2.40}$$

is called the *mass-based metric*. The mass-weighted metric will, loosely speaking, weight vertices by the local amount of mass belonging to a vertex. It essentially measures "how much mass needed to be moved" to go from deformation u to v. As mentioned previously, mass-based inner product and metric are very natural in linear modal analysis, since they make the linear modal matrix orthonormal:

$$U_{\text{lin}}^{(j)T} M U_{\text{lin}}^{(i)} = \delta_{ij}, \tag{2.41}$$

i.e.,

$$U_{\text{lin}}^T M U_{\text{lin}} = I_{3n}. \tag{2.42}$$

Given a set of deformations $u^{(1)}, u^{(2)}, \ldots, u^{(N)}$, dimensionality r, and a mass matrix M, the goal of mass-PCA is to find the r-dimensional hyperplane for which the sum of squared *mass*-projection errors in the mass metric is minimized. Note that we assume here that the hyperplane passes through the origin, since we want the zero deformation to be representable in the model. The projection with respect to mass differs from the standard orthogonal projection in $\mathbb{R}^{3n}$: if matrix U specifies some mass-orthonormal basis of some r-dimensional subspace of $\mathbb{R}^{3n}$, then the projection of a vector $x \in \mathbb{R}^{3n}$ onto the linear subspace spanned by U equals $UU^T M x$. The total mass-projection error is then

$$\sum_{i=1}^{N} \|u^{(i)} - UU^T M u^{(i)}\|_M^2.$$

The mass matrix of a deformable solid mechanics model is always symmetric and positive definite. Using Cholesky decomposition $M = LL^T$, it can be shown that substitution $z^{(i)} = L^T u^{(i)}$ translates the problem to a standard Euclidean PCA problem for the dataset $\mathcal{Z} = \{z^{(i)} \mid i = 1, \ldots, N\}$. Also, the resulting best Euclidean-orthonormal basis V for $\mathcal{Z}$ satisfies $V = L^T U$, where U is the optimal mass-scaled basis. To perform mass-scaled PCA, we explicitly form the $z^{(i)}$, and perform standard PCA on $\mathcal{Z}$. Mass-orthonormal basis U is then obtained by solving linear systems $L^T U = V$. Note that for models of constant mass density mass-scaled PCA becomes volume-scaled PCA. Overall, mass-PCA computation times were short for our simulations: on the order of

a few seconds for all of our models. As such, they do not represent a computational bottleneck of our precomputation; modal derivative and reduced polynomial coefficient computation times were significantly longer (on the order of two hours for models with say $20,000$ elements).

Cholesky decomposition of the mass matrix: Mass-PCA requires computing the Cholesky decomposition of the mass matrix, which is typically a large sparse matrix. The most obvious approach to do this is to use the standard complete Cholesky decomposition of the mass matrix. However, this can lead to fill-in of the Cholesky factor. The lower-diagonal Cholesky factor L will have non-zero entries everywhere that M does, plus at a certain number of fill-in locations. This number could potentially be large, potentially exceeding memory storage, or causing slow-downs when the L factor is used in the mass-PCA algorithm. One solution is to lump the mass matrix into a diagonal matrix, causing L to also be diagonal (often done in graphics). Another approach to decomposing the mass matrix is to first appropriately permute the vertices of the model, and then perform Cholesky decomposition on the permuted mass matrix. This approach avoids lumping and is just as accurate as the non-permuted Cholesky decomposition. We note that such permutations are standard, and have also been used to minimize memory requirements when timestepping the equations of motion using direct solvers. Figure 2.10 gives comparison of fill-in in the Cholesky factor of the non-lumped mass matrix for the bridge model (11,829 simulation vertices). Two strategies are compared: no permutation of the non-lumped mass matrix and permuting vertices using minimum degree ordering.

We also tried repeating the same experiment for larger models (Eiffel tower, heart, dragon), but the unpermuted Cholesky decomposition ran out of memory (2 Gb). Using minimum degree ordering, however, we were able to compute Cholesky factors for all these models. The number of non-zero entries in the mass matrix/Cholesky factor under minimum degree ordering was $0.031\%/0.140\%$, $0.022\%/0.147\%$, $0.011\%/0.059\%$, for the Eiffel tower (20,713 vertices), heart (28,041 vertices), and dragon model (53,449 vertices), respectively. Cholesky decomposition computation times were less than 1 minute in Matlab in all the cases. Minimum degree ordering times are fast (on the order of a few seconds). We also tried other permutation strategies, such as Reverse Cuthill-McKee [CM69, LS76], and a simple Column Count strategy (which sorts the rows and columns according to the number on nonzero elements), and found minimum degree ordering to give the smallest Cholesky factors. For more information on minimum degree ordering, see [GL89]. Another useful reference for the various permutation strategies is Matlab's help on sparse matrices.

2.8.2 Sketch-based basis generation

The idea here is to first construct an interactive *geometrically linear* model, such as a linear modal analysis model, and then introduce a human user into the loop, effectively using the linear model as a bootstrapping mechanism to obtain a basis of nonlinear deformations. The user first interacts with a linear vibration model [Sha90, JP02]. We use a static model to avoid displaying the dynamic deformation effects as those can sometimes provide more confusion than guidance to the user.

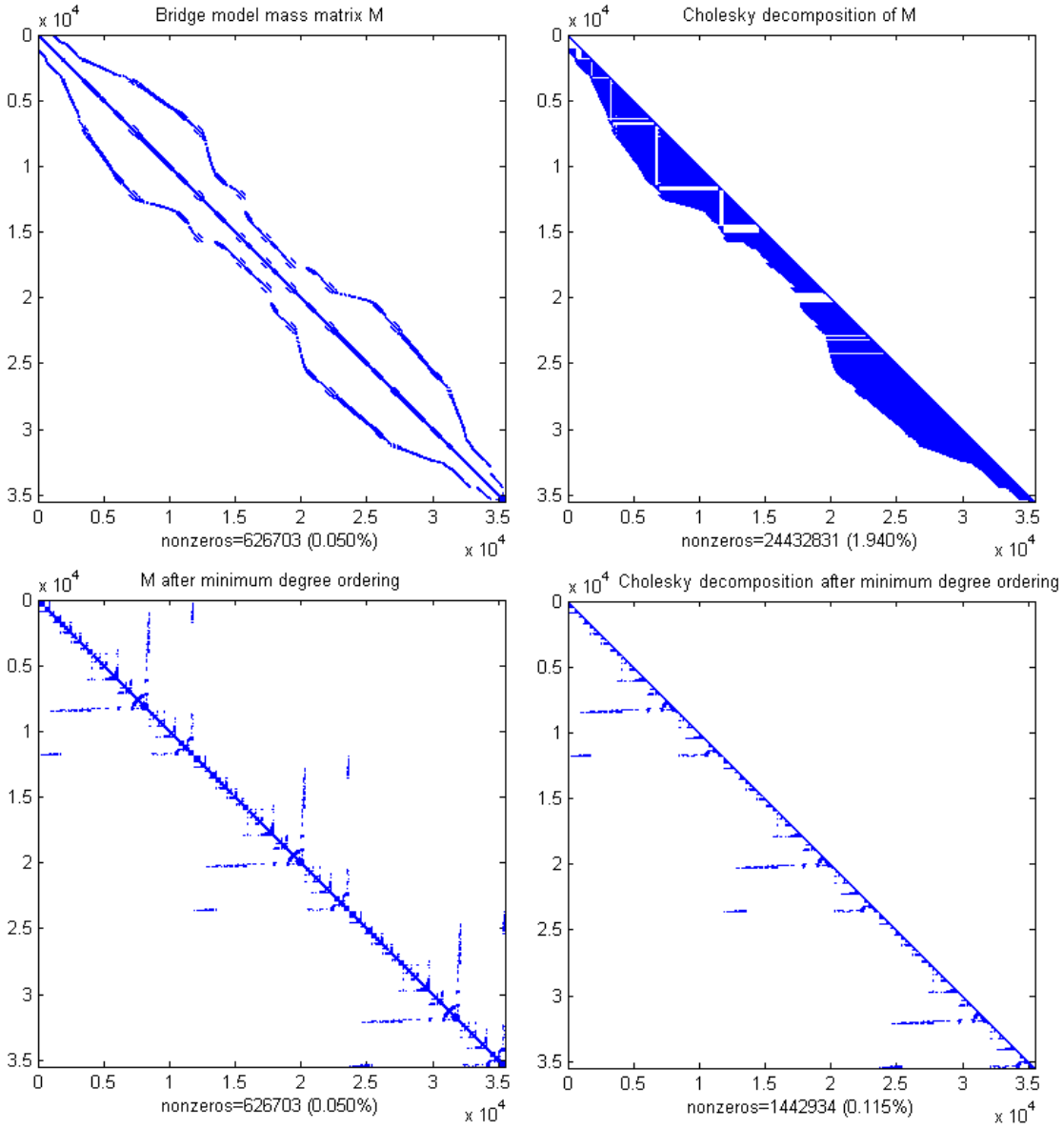


Figure 2.10: **Mass matrix Cholesky decomposition (bridge model, 11,829 simulation vertices):** Top-left: non-zero entries in the original mass matrix M (mesh generated via squashing cubes); Top-right: Cholesky factor of M; Bottom-Left: Permuted mass matrix (minimum degree ordering permutation); Bottom-Right: Cholesky factor of the permuted mass matrix.

Due to linearity, the model distorts badly for large deformations, but still provides a clue to the deformation involved. The forces imposed by the user are recorded to disk. A subset of these forces is automatically selected so that a certain separation distance is maintained among consecutive forces. This can be done with a greedy approach that iteratively removes force samples which are too close to other samples, with the removal prioritized based on local density of the samples

as defined under some radial kernel. The retained force samples are then sent as input to a full unreduced offline static solver. For every imposed load f, the static solver computes the static rest configuration u. Again, a subset of all deformations is automatically selected to maintain a certain separation mass-distance. Mass-PCA is then applied on the resulting shapes to extract the basis of motion U. When this basis is later used for an interactive nonlinear simulation, the model will be able to simulate nonlinear deformations similar to those sketched. Additional sketches can be used to refine the motion basis as desired. This approach can be seen as utilizing human intuitive ability to identify the "important" deformations, which a computer algorithm would have a very hard time identifying.

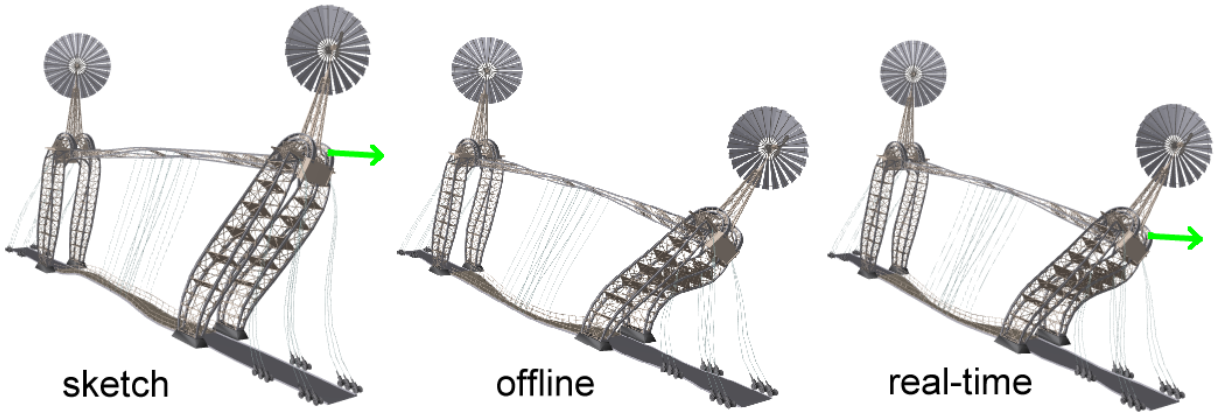


Figure 2.11: **Basis from Sketch:** Left: User interacts with a linear model. Resulting shape is distorted. Center: Applied force is recorded and sent to an unreduced offline static solver to solve for the corresponding nonlinear shape. Several such shapes are then processed by mass-PCA to obtain a basis of motion. Right: If same force is re-applied during the reduced runtime simulation, a shape which is visually almost indistinguishable from the center image emerges.

Model	num selected force loads	num selected deformations	static solve
spoon	353	45	45 min
bridge	326	142	2.4 hours

Figure 2.12: **Precomputation Timings for the Basis from Sketch.**

2.8.3 Basis from modal derivatives

Linear modal analysis [Sha90] provides a quality deformation basis for small deformations away from the rest pose. Intuitively, modal basis vectors are orthogonal directions into which the model can be pushed with the smallest possible increase in elastic strain energy. A generalization is possible: for any deformation $u_0 \in \mathbb{R}^{3n}$, *tangent linear vibration modes* give the best basis for

small deformations away from the deformation pose u_0. The first $k \geq 1$ tangent linear vibration modes at u_0 (denoted by $\Psi^i(u_0), i = 1, \ldots, k$) are the mass-normalized eigenvectors corresponding to the k smallest eigenvalues $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_k$ of the symmetric generalized eigenproblem

$$(K(u_0))x = \lambda M x. \tag{2.43}$$

Note that $K(u_0)$ and M are symmetric, M is always positive-definite and $K(u_0)$ is typically positive-definite for u_0 from some reasonably large neighborhood of the origin. Tangent linear modes coincide with linear modal analysis modes at the origin (define $\Psi_i := \Psi_i(0^{3n})$). Standard linear modal analysis simulation uses linear modes with *linear* forces and suffers from very visible errors for large deformations. A small improvement can be achieved by using $U = \{\Psi^i \,|\, i = 1, \ldots, k\}$ as a deformation basis in a reduced subspace integrator (i.e. with *nonlinear* reduced internal forces). In our experiments, we clearly detected a modest improvement over a linear simulation with linear modes. However, the linear modes are not really a good basis for large deformations, and the simulation tends to "lock" already for relatively small deformations. Locking (artificial stiffening) is discussed in Section 2.13.2.

In order to create a better basis, one can investigate how tangent linear vibration modes $\Psi^i(u_0)$ change with u_0. In [Nic76], they recompute the tangent linear vibration modes every certain number of timesteps along the solution path, which would result in periodically very slow frames if used interactively. Instead, we evaluate the directional derivative of $\Psi^i(u_0)$, at the origin, for the linear modal deformations of the form

$$u_0(p) = \sum_{\ell=1}^{k} \Psi^\ell p_\ell, \tag{2.44}$$

as shown in [IC85b]. We then use mass-PCA to generate the deformation basis U. Here, parameter $p = [p_1 \; p_2 \; \ldots]^T$ is the vector of *modal participation factors*. Linear modes $\Psi^i(u_0)$ can now be seen as a function of p (since u_0 is a function of p via Equation 2.44). The unnormalized *modal derivatives* can be defined as

$$\Phi^{ij} = \frac{\partial}{\partial p_j}\Big(\Psi^i\Big(\sum_{\ell=1}^{k} \Psi^\ell p_\ell\Big)\Big)_{|p=0^k}. \tag{2.45}$$

As in [IC85a], one can differentiate Equation 2.43 with respect to u_0, to show that Φ^{ij} are a solution to the following equation:

$$\Big(K - \lambda_i M\Big)\Phi^{ij} = \Big(M\Psi^i(\Psi^i)^T - I_{3n}\Big)\Big((H : \Psi^j)\Psi^i\Big). \tag{2.46}$$

If the inertia terms are neglected, derivatives are symmetric ($\Phi^{ij} = \Phi^{ji}$), and can be precomputed by solving the linear systems

$$K\Phi^{ij} = -(H : \Psi^j)\Psi^i, \tag{2.47}$$

where

$$H_{ij\ell} = \frac{\partial}{\partial u_\ell}\Big(K_{ij}(u)\Big)_{|u=0^{3n}}, \qquad i,j,\ell = 1,\ldots,3n \tag{2.48}$$

denotes the *Hessian stiffness tensor*. This third-rank tensor is the derivative of the stiffness matrix at the origin. It can be easily determined through analytical differentiation of formulas for the unreduced tangent stiffness matrix. The contribution of element e to blocks corresponding to vertices a, b, c of the full unreduced stiffness matrix and Hessian tensor at the origin, and the mass matrix (ρ is mass density) are:

$$M_e^{ab} = \Big(\int_e \rho\phi_a \cdot \phi_b dV\Big) I_3 \ \in \mathbb{R}^{3,3}, \qquad K_e^{ab} = A_1^{ab} + B_1^{ba} + A_2^{ba} \ \in \mathbb{R}^{3,3}, \tag{2.49}$$

$$\begin{aligned} H_e^{abc} = (C_1^{abc} + C_2^{bca} + C_2^{cba}) \otimes I_3 \quad + \quad I_3 \otimes (C_1^{cba} + C_2^{acb} + C_2^{bca}) \quad + \\ + \ \textstyle\sum_{i=1}^{3}\Big(e_i \otimes (C_1^{bca} + C_2^{abc} + C_2^{cba}) \otimes e_i\Big) \ \in \mathbb{R}^{3,3,3}. \end{aligned} \tag{2.50}$$

The contraction $H : a$ (for a vector $a = \sum_{\ell=1}^{k} a^\ell e_\ell$) denotes the matrix where element (i, j) equals $\sum_{\ell=1}^{k} H_{ij\ell} a^\ell$, for $i, j = 1, \ldots, 3n$. The system given by Equation 2.47 is a large sparse linear system of equations, with multiple right-hand sides to solve for. Therefore, we decompose K by the incomplete Cholesky factorization and then use the incomplete factors L, L^T as a preconditioner to solve the linear systems of Equation 2.47. As expected, preconditioning resulted in dramatic reductions in computation time; for example, the computation time of the heart model's derivatives was reduced from ten hours to twenty-five minutes. Normalized modal derivatives $\overline{\Phi}^{ij}$ are then obtained easily by mass-normalizing Φ^{ij}.

Intuitively, adding modal derivatives to the linear basis can be seen as adding quadratic terms to extend the linear modal analysis linear vector space into a $k-$dimensional paraboloid:

$$u(p) = \sum_{i=1}^{k} \Psi^i p_i + \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} \Phi^{ij} p_i p_j + O(p^3). \tag{2.51}$$

We will derive (and discuss) this relationship more in Section 2.8.4. Equation 2.51 suggests that the linear space spanned by all vectors Ψ^i and Φ^{ij} is a natural candidate for a motion subspace. It could be processed with mass-Gramm-Schmidt to obtain a mass-orthonormal basis [IC85b]. However, its dimension $k + k(k+1)/2$ quickly becomes prohibitive. Instead, we scale the derivatives according to the eigenvalues of the corresponding linear modes, i.e., we obtain the low-dimensional deformation basis by applying mass-PCA on

$$\Big\{\frac{\lambda_1}{\lambda_j}\Psi^j \mid j = 1,\ldots,k\Big\} \ \cup \ \Big\{\frac{\lambda_1^2}{\lambda_i\lambda_j}\overline{\Phi}^{ij} \mid i \le j;\ i,j = 1,\ldots,k\Big\}. \tag{2.52}$$

Scaling is necessary to put greater weight on dominant low-frequency modes and their derivatives, which could otherwise be masked by high-frequency modes and derivatives. Note that if mass-PCA was applied to linear modes only under this scaling, it would re-create the linear modal

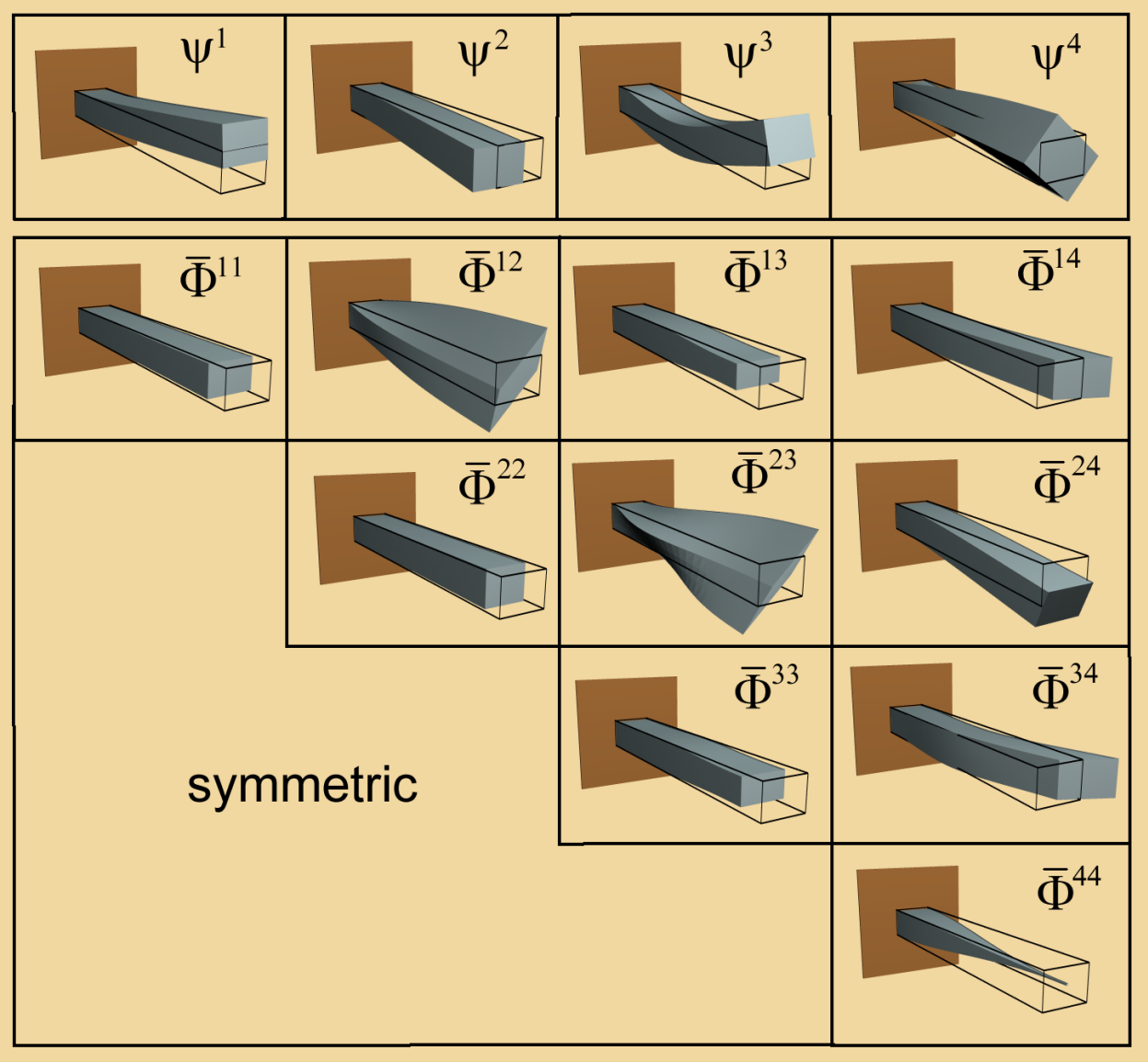


Figure 2.13: **Dominant linear modes and modal derivatives:** We exploit the statistical redundancy of these modes using mass-PCA of suitably scaled modes. All vectors are shown mass-normalized.

vectors with correct eigenvalues. The assignment of analogous weights to derivatives seems a natural generalization of the scaling. Note that K is a sparse symmetric matrix, and that different modal derivatives can be computed in parallel.

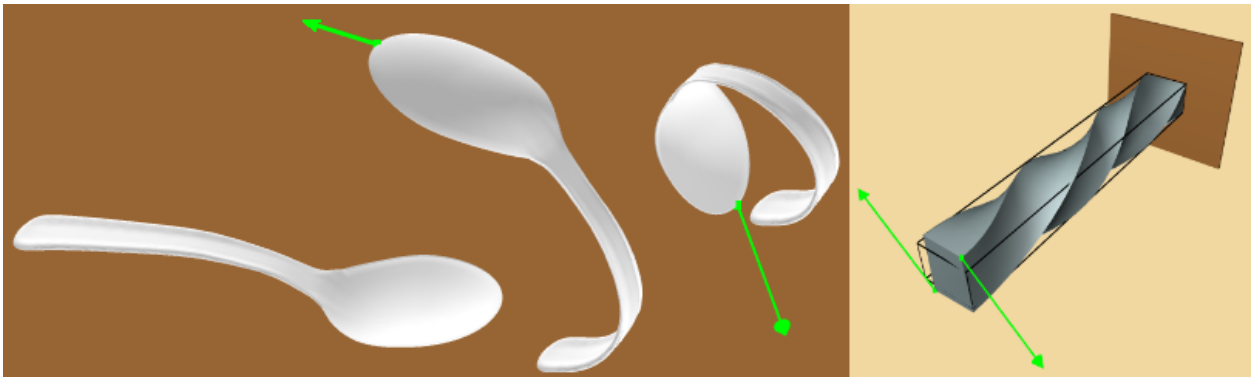


Figure 2.14: **Extreme shapes captured by modal derivatives:** Although modal derivative are computed about the rest pose, their deformation subspace contains sufficient nonlinear content to describe large deformations. Left: Spoon ($k = 6, r = 15$) is constrained at far end. Right: Beam ($r = 5$, twist angle=270°) is simulated in a subspace spanned by "twist" linear modes and their derivatives $\Psi^4, \Psi^9, \Phi^{44}, \Phi^{49}, \Phi^{99}$.

2.8.4 Alternative view on modal derivatives

Modal derivatives present one of the key concepts in this thesis. In this section, we give our alternative explanation of modal derivatives, different from the one in [IC85b]. This material is mathematically equivalent to material presented in Section 2.8.3. We provide this alternative interpretation to give more intuition about the derivatives.

Consider a deformable object which is sufficiently constrained so that it does not possess rigid degrees of freedom. For a static load f, the system will deform statically into deformation u, where u satisfies the unreduced static equation $R(u) = f$ (compare to Equation 2.4 on page 41). Because the gradient of R with respect to u, evaluated at the origin $u = 0$ (the stiffness matrix) is nonsingular, it follows from the implicit function theorem that the function R has an inverse in *some* sufficiently small neighborhood of the origin. That is, there exists a neighborhood $\mathcal{O}$ around the origin in the force space, such that for any force load $f \in \mathcal{O}$, there exists a unique deformation u such that $R(u) = f$. Consider what happens if we statically load the system into the direction of linear modes. In particular, suppose we apply a static force load $MU_{\text{lin}}\Lambda p$, where

Model	k	Compute linear modes	Build right-hand sides of Eq. 2.47	Solve Eq. 2.47
spoon	6	24 sec	6.5 sec	33 sec
tower	20	65 sec	226 sec	26 min
heart	20	111 sec	291 sec	28 min

Figure 2.15: **Computation of Modal Derivatives:** All performance data is given for a single 3.0 Ghz Pentium workstation with 2Gb of memory (manufactured in 2004). Mass-normalization and mass-PCA times were only a few seconds.

M is the mass matrix, $U_{\text{lin}} = [\Psi^1, \ldots, \Psi^k]$ is the matrix of mass-orthonormal *linear* modes, Λ is the diagonal matrix $\text{diag}(\lambda_1, \ldots, \lambda_k)$, and $p \in \mathbb{R}^k$ is some parameter that controls the strength of each mode in the load. In other words, these are the force loads which, for small deformations, produce deformations within the space spanned by the linear modes. Every value of p (from some sufficiently small neighborhood around the origin in $\mathbb{R}^k$) gives a load $MU_{\text{lin}}\Lambda p \in \mathcal{O}$, and therefore, there exists a unique deformation $u = u(p)$, such that

$$R(u(p)) = MU_{\text{lin}}\Lambda p. \tag{2.53}$$

Therefore, u is a well-defined function of p in some sufficiently small neighborhood of the origin. It is also C^∞ differentiable since $R = R(u)$ is C^∞ differentiable; therefore there exists a Taylor series expansion for u in terms of p. Can we compute (low-order) terms of this expansion? By differentiating Equation 2.53 with respect to p, one obtains

$$\frac{\partial R}{\partial u}\frac{\partial u}{\partial p} = MU_{\text{lin}}\Lambda, \tag{2.54}$$

which is valid for all p in the small neighborhood of the origin of $\mathbb{R}^k$. In particular, for $p = 0^k$, we get $K\frac{\partial u}{\partial p} = MU_{\text{lin}}\Lambda$ which is the expected linear modal response, $\frac{\partial u}{\partial p} = U_{\text{lin}}$. To compute the second order derivatives of u, we differentiate Equation 2.54 one order further by p, which, when we set $p = 0^k$, gives us

$$K\frac{\partial^2 u}{\partial p_i \partial p_j} = -(H : \Psi_j)\Psi_i. \tag{2.55}$$

Here, H is the Hessian tensor from Equation 2.48. By comparing Equations 2.47 and 2.55, it follows that $\frac{\partial^2 u}{\partial p_i \partial p_j} = \Phi^{ij}$, i.e., the second derivatives of $u = u(p)$ are precisely the modal derivatives. The second-order Taylor series expansion is therefore

$$u(p) = \sum_{i=1}^{k} \Psi^i p_i + \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} \Phi^{ij} p_i p_j + O(p^3). \tag{2.56}$$

The modal derivatives therefore span the natural second-order system response, for (large) deformations around the origin.

2.8.5 Modal derivatives generalization to second-order ODEs

The idea from the previous section is not limited just to ODEs originating from structural mechanics. It can be applied to general second-order systems on n ODEs of the form

$$A\ddot{x} + B\dot{x} + C(x) = f(t), \tag{2.57}$$

where $C = C(x)$ is a nonlinear function of $x \in \mathbb{R}^n$, A and B are some $n \times n$ matrices, and $f(t) \in \mathbb{R}^n$ is some time-dependent load. We assume A and

$$K = \frac{\partial C}{\partial x}_{|x=0} \tag{2.58}$$

to be symmetric positive-definite. One can define the linear modes just like in structural dynamics, by solving the generalized eigenvalue problem

$$K\psi = \lambda A\psi. \tag{2.59}$$

Note that this modal computation ignores the damping matrix B, which is a commonly used approximation [JP02, HSO03]; for greater accuracy, we could limit Equation 2.57 to $B = 0$. Next, one can define and compute the modal derivatives just like in the case of structural dynamics (Equations 2.47 and 2.48), using the Hessian (second derivative) of C. The interpretation of modal derivatives from Section 2.8.4 is still valid even in this generalized case. The linear modes and derivatives form a natural solution basis for the system of ODEs, since they give the natural second-order response to the system, under the natural (low-frequency) loadings of the system. This procedure can be seen as a priori model reduction for the system of ODEs from Equation 2.57, and is most suitable for ODEs where the solution exhibits oscillatory behavior around some rest configuration.

2.8.6 Comparison: modal derivatives vs sketch basis

The two bases essentially aim at different goals: modal derivatives try to correct the artifacts of linear modal analysis to produce a quality basis for global large deformations (without any presimulation data; a priori model reduction), whereas the sketch basis extracts a low-dimensional basis from given deformation data (a posteriori model reduction), so that similar deformations can be simulated in the future.

In practice, the modal derivative basis significantly extends the range of deformations that can be simulated as compared to the linear modal analysis basis, enabling simulations of much larger deformations. At the limit of this range, the deformations typically "lock": even if an even greater force is applied, the deformations do not change much. If the force load is removed, the model returns to the rest shape. For the cantilever beam model, for example, modal derivatives permit the beam to bend 90 degrees sideways, or twist up to about 270 degrees around its longitudinal axis. In the case of the bridge model, we found that the modal derivative basis "locks" when one of the masts is bent sideways to an angle of about 30 degrees from its original vertical location. The reason why the derivative basis eventually stops being a good basis under sufficiently large deformations is that the derivatives are a quantity computed around the origin (the zero deformation $u = 0^{3n}$): their computation involves the stiffness matrix *at the origin* and the gradient of the stiffness matrix *at the origin*. As such, the derivatives "do not know" how $R(u)$ and $K(u)$ behave under large deformations, except from what they can deduce from the knowledge of $R(0^{3n}), K(0^{3n}), (dK/du)(0^{3n})$. Of course, given some large deformation u, one could compute the linear modes and derivatives at u, and combine this basis with the basis at the origin. This could be done for a set of (appropriately chosen) samples $u^{(1)}, u^{(2)}, \ldots, u^{(N)}$, but we did not pursue this extension in this thesis, mainly since it is not obvious how to automatically select the samples and due to additional coding complexity. One possibility is that the samples could be specified (sketched) by the artist, by using a large deformation modal derivative basis (computed

at the origin, just as in Section 2.8.3) as a boot-strapping mechanism. Another possibility for basis extension would be to use a third-order model in Equation 2.56, leading to $O(k^3)$ *modal Hessians*. In addition to $K(0^{3n})$ and $(dK/du)(0^{3n})$, their computation would also involve $(d^2K/du^2)(0^{3n})$. However, note that if these bases are to be used for interactive simulation, the basis size must be kept moderate (up to about $r = 40$ on our hardware), because the evaluation of reduced internal forces becomes progressively expensive with larger basis sizes.

In this thesis we used a sketching basis when we wanted to go beyond the range of deformations supported by the modal derivatives at the origin. For example, a sketched bridge basis specialized to mast deflection was able to bend the mast to an extreme deformation of 90 degrees, making the top of the mast touch the ground. This example demonstrates than the sketch basis can sometimes achieve effects not captured by the modal derivative basis, provided that the user appropriately guides the force load selection to include the desired deformations into the basis.

The two bases in practice often overlap to some extent. For example, if the data presented to the sketch was generated under force loads similar to those causing linear mode excitation, then the sketch method will give a subspace similar to the derivative basis. Furthermore, such linear mode excitations tend to occur naturally, even if the user, say, just pulls on one vertex. The low-dimensional linear modes will get excited, together with some high-frequency deformations in the region of the application of the force. Such behavior is not surprising since the linear modes are the natural modes of the structure.

2.8.7 Basis for unconstrained models

Section 2.8.3 demonstrated how to determine modal derivatives for anchored meshes. However, models with no constrained vertices (unconstrained models) are very useful in computer graphics; many such objects can be, for example, combined into a flexible multibody dynamics simulation [Sha05] (see Figure 2.16). Models without constrained vertices require special care as the first six eigenvalues $\lambda_1, \ldots, \lambda_6$ are zero with the eigenvectors spanning the space of translations and infinitesimal rotations. Nevertheless, derivatives are still defined via Equation 2.45. To form a motion basis, we combine linear modes $\Psi^i, i \geq 7$ with derivatives $\overline{\Phi}^{ij}, i, j \geq 7$ (appropriately scaled, followed by mass-PCA). Rigid body motion can then be coupled with deformations [TW88, Sha05]. Note that this procedure does not guarantee deformations caused by inertia forces to be included in the motion basis (such deformations could be added explicitly; also, the first few non-rigid linear modes are sometimes similar to inertia-induced deformation shapes).

To compute the derivatives, first note that the approach from Equation 2.47 (page 67) is not directly applicable: stiffness matrix K is now singular (nullspace dimension is six), and there is no guarantee that Equation 2.47 has a solution. One approach to determine $\Phi^{ij}, i, j \geq 7$ is to use the derivative formulation of Equation 2.46, that is, the formulation where the mass terms were not neglected:

$$\Big(K - \lambda_i M\Big)\Phi^{ij} = \Big(M\Psi^i(\Psi^i)^T - I_{3n}\Big)\Big((H : \Psi^j)\Psi^i\Big). \tag{2.60}$$

Note that this causes modal derivatives to no longer be symmetric: $\Phi^{ij} \neq \Phi^{ji}$. The matrix $K - \lambda_i M$

is singular (its nullspace consists of multiples of Ψ_i), but the equation still has a solution (it was obtained by taking gradients of $K(p)\Psi_i(p) = \lambda_i(p)M\Psi_i(p)$ with respect to p, so by construction there always is a solution). To find a solution, one can solve the regularized version of the system, obtained by replacing $K-\lambda_i M$ with $\overline{K} := K-\lambda_i M+\Psi^i(\Psi^i)^T$. Note that $K-\lambda_i M$ is symmetric, and hence

$$\mathrm{im}\Big(K-\lambda_i M\Big) = \Big(\ker(K-\lambda_i M)\Big)^{\perp} = \Big(\mathrm{Lin}(\{\Psi^i\})\Big)^{\perp}, \tag{2.61}$$

where $\mathrm{im}(\_)$ and $\ker(\_)$ denote the range and nullspace of a matrix, respectively. Then, if $x = y + \alpha\Psi_i$ solves $\overline{K}x = b$ (assuming $b \in \mathrm{im}(K-\lambda_i M)$, $y^T\Psi_i = 0$, and $\alpha \in \mathbb{R}$), it follows that $\alpha = 0$ and $(K-\lambda_i M)x = b$.

Any multiple of Ψ_i can be added to any solution of Equation 2.46. A particular solution Φ^{ij} can be chosen by imposing $(\Psi^i)^T M\Phi^{ij} = 0$, i.e., the derivative is made mass-orthogonal to its linear mode. The matrix $\overline{K}$ is not sparse and will often have negative eigenvalues [1]. However, the "black-box" multiplication $x \mapsto \overline{K}x$ can be performed efficiently (by computing $Kx-\lambda_i Mx+\Psi_i(\Psi_i^T x)$), and used in a fast sparse symmetric (since $\overline{K}^T = \overline{K}$) solver, such as MINRES.

2.9 The implicit Newmark (subspace) integrator

To timestep the simulation at runtime, we numerically integrate the system from Equation 2.21, which is a nonlinear system of r coupled second-order differential equations. Nonlinearity occurs due to the forcing and damping terms. We use the *implicit Newmark integrator* (see [KLM01, Wri02]), which is popular in structural dynamics. It is second-order accurate in terms of local error: so if the timestep is h, the error of a single step is on the order of $O(h^2)$.

One implicit Newmark timestep essentially consists of solving the current *dynamic equilibrium* equations of motion, so that the sum of mass inertia forces, damping forces, and internal forces balance the currently applied external forces. This equilibrium is found by solving a nonlinear r-dimensional equation of the form $F(q+\Delta q) = 0^r$, where q are the reduced coordinates before the timestep, and Δq is the change in reduced coordinates during this timestep (the unknown). The nonlinear $r-$dim vector function F changes from timestep to timestep. The equation $F(q+\Delta q) = 0^r$ is solved by performing a series of Newton-Raphson iterations [Wri02], constructing consecutive approximations $q+(\Delta q)^{(1)}, q+(\Delta q)^{(2)}, \ldots$, analogous to a Newton solver to solve a 1D nonlinear equation. Each of these iterations requires constructing a certain system matrix A of a dense $r \times r$ linear system, constructing a certain r-dim dimensional right-hand side vector b, and then solving $Ax = b$. Both A and b of course change from iteration to iteration. The solution approximation is then updated as $(\Delta q)^{(j+1)} = (\Delta q)^{(j)} + x$. The linear system to be solved is a dense $r \times r$ symmetric linear system, and we solve it using a direct symmetric matrix solver. Note that iterative solvers are not as attractive in our case due to relatively small r and dense

[1] There is no simple formula for its eigenvalues; however, the spectrum of the generalized eigenvalue problem $\overline{K}x = \lambda Mx$ equals the spectrum of $Kx = \lambda Mx$, shifted by $-\lambda_i$, and with the resulting zero eigenvalue changed to 1.

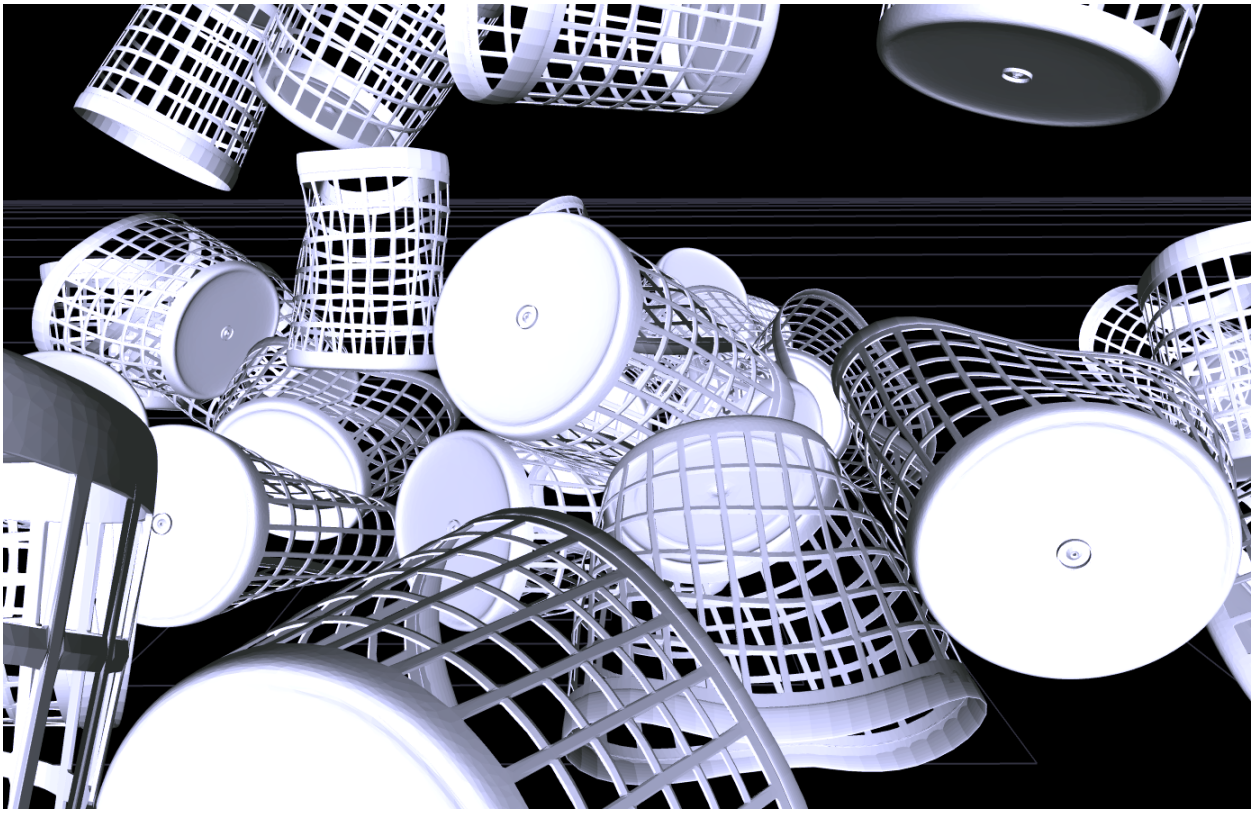


Figure 2.16: **Multibody dynamics simulation with large deformations:** In this simulation, we dropped 512 flexible baskets to the ground. The baskets started from zero velocity, horizontally separated (but in close proximity) at a certain height. Collision detection was performed using a BD-Tree [JP04]. Penalty forces were applied between leaf spheres to resolve contact. Motion basis ($r = 40$) used linear modes $\Psi_7, \ldots \Psi_{26}$ and their derivatives. The rigid body motion was not coupled with deformations, although this would not be a difficult extension [Sha05]. It took 1.2 sec total to perform the 512 implicit Newmark steps to advance the reduced deformations to the next graphical frame.

matrices. Constructing A and b requires evaluating the reduced tangent stiffness matrix $\tilde{K}$ and reduced internal forces $\tilde{R}$ in configuration $q + (\Delta q)^{(j)}$. The implicit Newmark integrator is given as follows.

Algorithm *One step of implicit Newmark subspace integration*
Input: values of $q, \dot{q}, \ddot{q}$ at timestep i, reduced external force $\tilde{f}_{i+1}$ at timestep $i + 1$; max number of Newton-Raphson iterations per step $j_{\max}$ (semi-implicit solver: $j_{\max} = 1$); tolerance TOL to avoid unnecessary Newton-Raphson steps; timestep size Δt.
Output: values of $q, \dot{q}, \ddot{q}$ at timestep $i + 1$
1. $q_{i+1} \leftarrow q_i$;
2. **for** $j = 1$ **to** $j_{\max}$ // perform a Newton-Raphson iteration:

3. Evaluate reduced internal forces $\tilde{R}(q_{i+1})$;
4. Evaluate reduced stiffness matrix $\tilde{K}(q_{i+1})$;
5. Form the local damping matrix
6. $\tilde{C} = \alpha\tilde{M} + \beta\tilde{K}(q_{i+1})$; // in our work $\tilde{M} = I_r$
7. Form the system matrix $A = \alpha_1\tilde{M} + \alpha_4\tilde{C} + \tilde{K}(q_{i+1})$;
8. residual $\leftarrow \tilde{M}(\alpha_1(q_{i+1} - q_i) - \alpha_2\dot{q}_i - \alpha_3\ddot{q}_i)+$
9. $+\tilde{C}(\alpha_4(q_{i+1} - q_i) + \alpha_5\dot{q}_i + \alpha_6\ddot{q}_i) + \tilde{R}(q_{i+1}) - \tilde{f}_{i+1}$;
10. **if** ($\|\text{residual}\|_2 < TOL$)
11. break out of for loop;
12. Solve the $r \times r$ dense symmetric linear system:
13. $A\Big(\Delta q_{i+1}\Big) = -\text{residual}$
14. $q_{i+1} \leftarrow q_{i+1} + \Delta q_{i+1}$;
15. $\dot{q}_{i+1} \leftarrow \alpha_4(q_{i+1} - q_i) + \alpha_5\dot{q}_i + \alpha_6\ddot{q}_i$; // update velocities
16. $\ddot{q}_{i+1} \leftarrow \alpha_1(q_{i+1} - q_i) - \alpha_2\dot{q}_i - \alpha_3\ddot{q}_i$; // update accelerations
17. Return $q_{i+1}, \dot{q}_{i+1}, \ddot{q}_{i+1}$;

Integrator uses parameters $0 \leq \tilde{\beta} \leq 0.5,\ 0 \leq \tilde{\gamma} \leq 1$, and constants

$$\alpha_1 = \frac{1}{\tilde{\beta}(\Delta t)^2},\ \alpha_2 = \frac{1}{\tilde{\beta}\Delta t},\ \alpha_3 = \frac{1-2\tilde{\beta}}{2\tilde{\beta}},\ \alpha_4 = \frac{\tilde{\gamma}}{\tilde{\beta}\Delta t},\ \alpha_5 = 1 - \frac{\tilde{\gamma}}{\tilde{\beta}},\ \alpha_6 = \Big(1 - \frac{\tilde{\gamma}}{2\tilde{\beta}}\Big)\Delta t.$$

We chose $\tilde{\beta} = 0.25$, $\tilde{\gamma} = 0.5$, which is a common setting for many applications. Explicit central differences integration is defined by $\tilde{\beta} = 0, \tilde{\gamma} = 0.5$. Constants α, β are Rayleigh damping constants. Parameter β (*stiffness-proportional damping*) improves the stability of the stimulation as it has the effect of damping high frequencies of motion, which are often just numerical noise. Parameter α (*mass-proportional damping*) controls the "under-water" damping effect. In some cases we set α to zero, as stiffness-proportional damping is sufficient.

We note that for a general nonlinear material, $\tilde{R}(q)$ is a complicated function. For a general (i.e., nonlinear) isotropic hyperelastic material, it is a large sum of rational functions involving logarithmic terms. In general, it has several poles, and does not possess an immediate compact and simple analytical expression. Hence, direct evaluation of such functions is non-trivial. One could proceed by evaluating full unreduced forces $R(Uq) \in \mathbb{R}^{3n}$ and forming its explicit projection $\tilde{R}(q) = U^T R(Uq)$ (and similarly for the reduced tangent stiffness matrix), however such an approach is currently not real-time for large models.

As common in computer graphics, we found it sufficient to perform a single Newton-Raphson iteration per timestep ($j_{\max} = 1$). This is a speed-accuracy tradeoff, and if necessary, multiple Newton-Raphson iterations can be performed per timestep. If the timestep is set too large, a single Newton-Raphson iteration will not be sufficient to maintain a convergent solution, and can lead to the solution "blowing up". Also note that even if an arbitrary number (that is, as many as necessary to reach convergence) of Newton-Raphson iterations is performed per timestep, convergence is not guaranteed: large timesteps or drastic changes in external forces can cause the iteration to start too far away from the solution, and then the iteration procedure will not converge. This is similar to

how a 1D Newton solve is not guaranteed to always find a root of a non-convex 1D function. In practice, however, we were always able to pick timestep values that were stable and were at the same time large enough to reach real-time performance.

2.9.1 Why not explicit subspace integration?

When developing this research, we first tried using the central differences integrator, which is an explicit integrator from the Newmark family, commonly used in explicit finite element commercial codes such as Abaqus Explicit. Explicit integration has the convenient advantage that it is no longer necessary to solve any dense $r \times r$ linear systems per timestep. Only the reduced internal forces are necessary to progress the solution to the next timestep. The reduced tangential stiffness matrix is no longer required for the integrator; it is, however, still required for the tangential Rayleigh damping. Unfortunately, the explicit simulations turned out to be extremely unstable with the typical timesteps in our simulations, and it wasn't until we switched to the implicit Newmark integrator that we were able to produce useful results. In this section, we will explain why this was the case.

First, note that linear modal analysis essentially band-limits the solution to low frequencies. High-frequency content is removed from the solution, and as such, it is readily possible to timestep linear modal analysis simulations with an explicit integrator. This property, however, does not carry over to our nonlinear reduced simulations in a large-deformation motion basis. The reason for this is the presence of high frequency components in reduced simulations (see Table 2.1). Namely, if one deflects an object into its first linear mode, its modal derivative will enter the solution via nonlinear coupling of the basis vectors. This derivative is some very stiff shape, and as such its natural oscillation frequency is very high. Therefore, if the system were simulated in a basis consisting of the linear mode and its derivative, the resulting system will have a very high frequency for small oscillations around the origin. The situation is similar if a basis consists of several linear modes and their derivatives, or even if the motion basis is obtained from a sketch (see reduced spectra in Table 2.1). If an explicit integrator is used, the timestep must be sufficiently small to resolve the period of the highest frequency present in the system, which typically forces the timestep to be very small. In contrast, if an implicit solver is used, then the oscillations corresponding to frequencies higher than the Nyquist frequency of the timestep will be numerically damped. The stiff high-frequency shapes will appear in the solution (counteracting the artifacts of linear modes), but their own inherent oscillatory dynamics will be suppressed.

If explicit integration is to be used nonetheless, how to pick a timestep that will be stable? Stability for *small deformations* can be determined by solving the *dense* $r \times r$ generalized eigenvalue problem $\tilde{M}x = \lambda\tilde{K}(0^r)x$ (typically, $\tilde{M}$ is identity in our simulations due to mass-normalization of modes, giving a standard eigenvalue problem). The square root of the largest eigenvalue of this system is the largest natural frequency (in radians/sec) of the reduced system (call it ω.) If the explicit timestep is chosen to be less than $2\pi/\omega$ (Courant-Friedrichs-Lewy (CFL) condition), the simulation will be stable in the vicinity of the rest pose. Stability of the nonlinear system at the origin of course does not guarantee global stability. As the simulation moves toward large defor-

Spectrum case	Eiffel tower		Heart	
	min	max	min	max
linear spectrum ($k = 20$), in Hz	1.0	15.5	1.0	5.6
spectrum of the reduced system ($k = 20, r = 30$), in Hz	1.0	517.0	1.0	70.8
spectrum of the unreduced system, in Hz	1.0	11811	1.0	21986

Table 2.1: **Reduced systems have high frequency spectra:** Linear spectrum row gives the range of the first 20 natural linear modal frequencies of the structure. For both the Eiffel tower and the heart model, the Young's modulus was chosen such that the lowest frequency is 1 Hz, so that frequencies are in an interesting range for computer graphics; but the same properties carry over to stiffer structures with higher frequency ranges. The reduced spectrum row gives the range of the frequencies of the linearized *reduced* system, i.e., the range of $\sqrt{\lambda}/(2\pi)$, where λ traverses the eigenvalues of the reduced tangent stiffness matrix $\tilde{K}$ at the origin. The reduced system was obtained by augmenting the first $k = 20$ linear modes with their derivatives, and mass-PCAing this assembly of vectors to a $r = 30$ dimensional basis. For comparison, we also give the spectrum of the unreduced FEM model (last row).

mations, the spectrum of $\tilde{K}(q)$ changes. Typically, the structure becomes stiffer and the maximum eigenvalue typically increases; therefore, smaller explicit timesteps will be required.

Because stability is very important for interactive applications, and because local Rayleigh damping model requires the assembly of the reduced tangent stiffness matrix anyway, we use the implicit integrator in all of our simulations. The simulations are stable and fast. With bigger values of r (e.g., $r \geq 15$), the majority of the simulation time is spent evaluating the reduced internal forces and the reduced tangential stiffness matrix, both an $O(r^4)$ operation, in contrast to the $O(r^3)$ dense system solve. For this reason, avoiding a system solve via an explicit method does not accelerate the simulation significantly. Also, performing many very small explicit timesteps is impractical, due to the $O(r^4)$ reduce force evaluation cost for each timestep.

2.9.2 Numerical damping

Unfortunately, the process of removing the unwanted frequencies (i.e., frequencies larger than what the timestep can represent) is not perfect with the implicit integrator. Inevitably, the process "spills over" to the frequency range which we do want to preserve in the simulation, resulting in a (visible) loss of energy in the system called *numerical damping*. Essentially, numerical damping is a form of error in the solution to the system of ODEs, and occurs even if Rayleigh damping is turned off. The true analytical solution to the reduced system of ODEs (without Rayleigh damping) exhibits no energy loss. The problem of numerical damping is well known [HW04b] and intrinsic to implicit integration. It is not specific to reduced models; for example, similar issues occur with implicit cloth solvers [BW98].

So what can be done to prevent numerical damping? First, the issue applies mostly to systems which are meant to be underdamped or not damped at all. In many engineering applications,

numerical damping is tolerated, since real structures always exhibit some amount of damping. If undamped systems are to be simulated, the solution is to make the timestep smaller. In this way, the reduced system of ODEs is solved more accurately, resulting in less numerical damping.

2.9.3 Choice of timestep for implicit Newmark

The quality of the chosen timestep is not an absolute quantity on the time scale. It depends on the ratio of the timestep to the periods of the fundamental frequencies of the system. Namely, the natural frequency spectrum of every FEM deformable object can be scaled by a constant simply by scaling the internal forces by an appropriate constant. In our simulations, if the Young's modulus is uniformly scaled by a constant $\gamma > 0$, all internal forces get multiplied by γ, and as a result, all natural frequencies are scaled by $\sqrt{\gamma}$. Effectively, this means that time is rescaled by $1/\sqrt{\gamma}$. It can be shown mathematically that if one introduces a substitution $\tau = (1/\sqrt{\gamma})t$ into the equations of motion, one obtains an ODE that *is* the equations of motion in τ for a deformable object that differs from the original one only by a linear scaling of Young's modulus, Rayleigh damping parameters, and mass density. A Newmark discretization with a timestep Δt for the original system is equivalent to a timestep $\Delta\tau = (1/\sqrt{\gamma})\Delta t$ for the new system. Note that the ratio of timestep versus the natural frequencies of the system is preserved.

Therefore, the accuracy of the integrator depends on the number of timesteps *relative to the natural frequency spectrum of the system*, such as, for example, the ratio of the timestep to the the lowest natural (linear) period of oscillation. The larger the number of these timesteps, the more accurate the solution to the reduced ODE. If the number of timesteps per period of a natural system frequency is too small, excessive numerical damping results: the frequency in question is damped quickly and is lost from the solution. Consequently, smaller timesteps are necessary to accurately simulate stiff structures with high natural frequencies .

For interactive simulation, the simulation time is often synchronized with the real time, such that if 1 second of real simulation time elapses, the numerical integrator will have progressed the solution forward by 1 second of simulation time. If a graphic frame rate of 30 Hz is maintained, at least 30 timesteps will be performed per second. In our experiments, this sufficiently avoided numerical damping for frequencies in the 1 Hz range (up to a few Hz). Performing more timesteps per second is of course beneficial to higher frequencies. For example, in our haptic simulation we perform 1000 timesteps per second, which adequately resolved frequencies up to several tens of periods per second, such as 10-30 Hz. Note that these are frequencies that are already very difficult to see visually, both due to the imperfections of the human visual system and due to a limited graphics update rate. For example, 30 Hz is the highest frequency that is not visually aliased on a 60 Hz graphical display.

2.10 External forces

The external forces in our simulation originate either from a user pulling interactively on a certain vertex (or a set of vertices) with a mouse, or from collision detection and contact force computations described in the second part of the thesis. The external forces can be conceptually organized into a $3n$-dimensional vector f, similar to the deformation vector u. The external forces are projected into the basis U by equation $\tilde{f} = U^T f$. The user interaction vector f is typically sparse, so this basis projection will typically be a sparse matrix-vector multiplication. Even if it were a dense multiplication, the execution time would be $O(nr)$, on par with the $O(nr)$ time necessary to evaluate $u = Uq$ to display the object on the screen.

2.11 Runtime modification of material parameters

Our method allows for runtime modification of mesh material parameters. Exact polynomials for the new values of material parameters can be generated interactively, since Lamé coefficients λ, μ and mass density appear linearly in the formulas for internal forces and the mass matrix. The mesh needs to be divided into separate groups, with constant material parameters over each group. Two polynomials are precomputed for each group, one collecting only the λ-terms (and setting $\lambda = 1$), and one involving only the μ-terms (and setting $\mu = 1$). To edit parameters, polynomials for each group are weighted by current group values of λ, μ, and all the group polynomials are summed together to produce the exact global polynomials. Changing mass density for different parts of the mesh can be done in a similar fashion. Also, the scheme could be extended to support a fixed material parameter distribution $\lambda = \lambda(X), \nu = \nu(X)$ over each group (as opposed to constant). Distributions can then be easily scaled to $\alpha\lambda(X), \beta\nu(X)$ by modifying per-group scaling constants α and β. Note that the precomputed basis will become less optimal if material parameters deviate too far from those used for precomputation. It can however be shown that the modal derivative basis is invariant under uniform global scaling of Young's modulus and/or mass density. Also, it is possible to omit any subset of basis vectors from the basis before each individual runtime invocation: the terms corresponding to omitted dimensions simply need to be dropped from the polynomials. In particular, any first $r' \leq r$ basis vectors can be used for a particular runtime invocation.

2.12 Graphical rendering

Although reduced dynamics can be integrated entirely in the subspace, we must reconstruct the full $3n$-dimensional displacement vector u before each rendering step (say at 30 Hz or 60 Hz), by performing matrix-vector multiply $u = Uq$. This multiplication can be easily performed on CPU. We have also implemented it in graphics hardware, which leaves more room on CPU for other computations (such as Newmark integration). One additional benefit of the GPU simulation is that

the model geometry is now effectively static and can be efficiently cached in a display list, which avoids bus-bandwidth bottlenecks of rendering dynamic deformable geometry.

2.12.1 Computing u=Uq on the GPU

Our GPU $u = Uq$ computation is an example of a GP-GPU (General Purpose GPU) simulation. It uses pixel buffers (usually referred to as "pbuffers"), render-to-texture, and vertex texture fetch capabilities of modern graphics cards to perform the computation. Previously, GPU implementations were already demonstrated in [JP02], but were performed by passing vertex modal displacements as vertex parameters to the *vertex* shader, which then assembled the vertex deformation. Since the number of vertex parameters is limited on modern GPUs, this in practice previously limited r to about 16, at least with the current generation of graphics cards. In our new approach, there are no explicit limitations on r; the only limitation is the size of texture memory which needs to store the basis matrix U.

Pbuffer is essentially a non-visible rendering buffer which resides in non-visible framebuffer memory. It can be used, for example, for off-screen OpenGL rendering, such as to render a large poster at a resolution higher than what supported by the monitor. Render to texture refers to rendering the scene into a texture, as opposed to into the visible part of the framebuffer. This texture can then be used in the next rendering passes/frames just like any other texture. Modern GPU cards support render to texture by rendering the scene into an appropriate pbuffer. *Vertex texture fetch* refers to fetching vertex-specific information from the GPU in the vertex shader, as opposed to the fragment shader. Traditionally, GPU texture memory access was only possible in fragment shaders; vertex texture fetches have been available (on Nvidia graphics cards) since 2005. Similar capabilities also exist on ATI's hardware; we used Nvidia's graphics cards in our work.

Our rendering algorithm works as follows. Matrix U is stored in texture memory. We found 16-bit floating point format to be sufficient. In pass 1, the vertex shader does nothing, while a fragment shader multiplies $u = Uq$ and renders the resulting deformation vector u to texture. Pass 1 is invoked by rendering a full-screen quad. Each pixel corresponds to one vertex of the model: the fragment shader fetches the corresponding $3 \times r$ subblock of U from texture memory, and then uses q (stored in a global parameter array) to compute the deformation of the vertex. The computed 3D deformation is the computed pixel's "color"; normally this color gets stored into the visible part of the framebuffer; but it in this case, the render target has been changed to a pbuffer texture, so effectively the "color" gets stored into a texture. Pass 2 is similar to a standard rendering pass, except each vertex's position is deformed in the vertex shader (in addition to projection/modelview matrix transformations), by fetching the vertex deformation from texture memory, and displacing the position of the vertex accordingly. This procedure is similar to how displacement maps could be applied (in general, not in our thesis) with vertex texture fetches in the vertex stage of the pipeline. A standard rendering pipeline follows: the vertex is lit, following by a potential application of a standard texture map in another fragment shader in pass 2.

With the most recent GPU cards that support *framebuffer objects*, the pbuffer and render-to-texture extensions are no longer necessary, as framebuffer objects can perform the same render-

to-texture functionality faster and with a cleaner user interface. However, this solution is not portable to graphics cards that predate (approximately) mid-2006. We note that another approach for accelerating the reconstruction of u would be to *compress* the basis shape vectors (columns of U), similarly to how Green's functions were compressed in [JP03].

Dynamic normals on the GPU

One issue with the GPU approach as presented this far is that the vertex normals are not updated with the deformation, which leads, for example, to incorrect (static) specular highlights when the object is deforming. Normals can be updated in graphics hardware, in a manner similar to deforming vertex positions. This is performed by pushing the original normals forward by the current local deformation gradient. Such a normal correction amounts to multiplying a certain vertex-dependent constant $3 \times r$ matrix with q, followed by a normalization of the corrected normal. These matrices can be assembled into a constant matrix U_N, which is stored in texture memory just like U. This same normal correction approach is also used to provide dynamic contact normals in our contact computation algorithm (second part of thesis). We discuss the approach and give exact formulas in Section 3.6.2. This procedure gives good approximations to the true normals when deformations consist of large rotation and small strain; it is, for example, not accurate in the presence of shear. Normals can be improved using the alternative approach of [RJ07].

2.12.2 Adaptive $u = Uq$ computation for collision detection

The deformation vector u is necessary if the simulation is to perform any collision detection. If a general collision detection scheme is used, and a CPU computation is not desirable, u can be obtained by a GPU computation followed by a GPU read-back operation. GPU read-backs are known to be slow and are often reported as a bottleneck in interactive simulations. In our approach, we use reduce-coordinate collision detection, which only requires knowledge of the reduced deformation vector q, and not of the unreduced deformation vector u. More precisely, the actual displacements of a certain set of vertices will still be required (e.g. in the case of a BD-Tree, vertices at the coarse hierarchical levels, and those close to contact), but this set is typically very small (and changes from timestep to timestep). Deformations for these vertices can be simply synthesized on the CPU. As such, we can run the deformable object dynamics *and* collision detection at haptic rates on the CPU, and only reconstruct $u = Uq$ at graphic rates for rendering, in the GPU pixel shader, without any need for GPU read-backs. We will describe this process in detail in the second part of the thesis.

2.13 Experiments

We present two experiments: one that compares different basis generation methods with the same basis size ($r = 12$), and one that investigates the modal derivative basis under progressively larger values of r. Our webpage gives video clips for all the experiments in this section [JJ].

2.13.1 Re-simulation, modal derivatives, sketch

In Figure 2.3 on page 46 we compared reduced simulations to an unreduced implicit Newmark simulation with full internal force and stiffness matrix computation. Same simulation parameters were used in all cases. Using a reduced interactive model, we recorded a short user-exerted vertical external force impulse, applied at the end of the spoon. This impulse was used to generate all the simulations, and was strong enough to push the spoon deeply into the nonlinear region. We simulated the reduced spoon in three $r = 12$ dimensional bases: a modal derivative basis (originating from $k = 6$ linear modes), a sketch basis (the user sketched upward motion of the spoon and some sideways motion), and a basis obtained by applying mass-PCA on the deformation trajectory of the unreduced simulation. As expected, in this last case ("re-simulation"), the resulting basis is close to optimal and the resulting reduced trajectory lies very close to the original motion. Of course, the basis is not available until *after* we have invested the time in performing the unreduced simulation, so the utility of such a basis is limited to re-simulation with small changes in applied force loads or mesh material parameters.

At around the first deformation maximum, a short transient wave motion occurs in the full solution and such traveling localized deformations are difficult to capture by subspace dynamics. The modal derivatives and sketching bases produce almost correct amplitudes and 4.6%, 10.1% smaller nonlinear frequencies, respectively. The modal derivative basis lacks high-spatial-frequency localized deformation, so it cannot capture (non-standing) deformation waves. The sketch basis was obtained under force loads less severe as the one used for the experiment, which we speculate to be an important source of error for the simulation in the sketched basis.

2.13.2 Modal derivative basis under progressive r

We performed this experiment on the voxelized dragon model (see Figure 2.17), with voxelization resolution of 120 x 120 x 120, yielding the most detailed model in this thesis (number of voxels = 27,314, number of voxel vertices = 53,449). The dragon is anchored to the ground at the bottom. The interior is hollow; if we had flood-filled the interior with voxels, this would result in 76,288 voxels and 91,261 voxel vertices, respectively. The stiffness matrix (in double precision) could then no longer be factored within the memory constraints of our system; adaptive (say, tetrahedral) meshing techniques could help [LS07].

We manually generated two concentrated force loads, one acting at the tip of the dragon's mouth, and one on top of the dragon's back. The force loads were distributed over a short time interval: 10 timesteps for the dragon's mouth and 30 timesteps for the dragon's back. We then computed the dynamic deformations under the unreduced nonlinear simulation, a fully linear simulation, and reduced nonlinear simulations in derivative bases with progressive values of r (see Figures 2.18, 2.19, 2.20, 2.21). The fully linear simulation gives the solution to the linear system of ODEs $M\ddot{u} + D\dot{u} + Ku = f$, and can capture local detail, but exhibits well-known volume-growth artifacts under large deformation. Note that a linear modal analysis simulation with the first k modes (not shown in this experiment) is an approximation to this linear model. Such a simulation

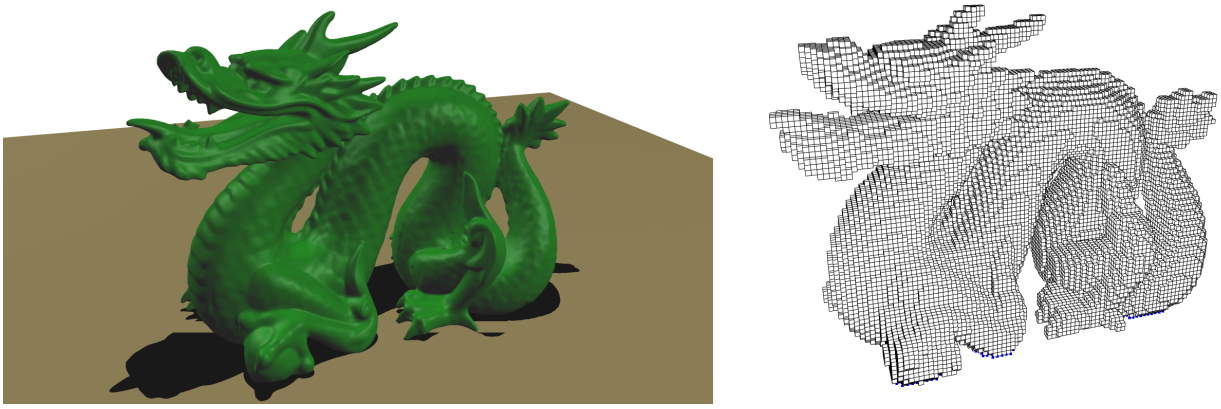


Figure 2.17: **The dragon model:** Left: the rendered dragon model in the rest configuration. Right: simulation voxel mesh. Bottom vertices are fixed rigidly.

would exhibit similar levels of volume growth, and would additionally be unable to simulate local deformation detail. Most reduced simulations use derivatives obtained from a basis with $k = 20$ linear modes. Maximum basis size is $r = 20 + 210 = 230$ in this case (linear modes and all derivatives concatenated into a basis, followed by a Gramm-Schmidt mass-orthogonalization). Bases for lower values of r were obtained using mass PCA, as described in Section 2.8.3.

All simulations use the same timestep of 1 / 300 sec, and identical implicit Newmark integration parameters. Video playback is 15x slower than actual time, and the entire simulation consisted of 90 ("head" force load) or 180 ("back" force load) implicit Newmark timesteps. As argued in Section 2.9.3, time re-scaling does not affect simulation accuracy. What matters is the number of timesteps per typical simulated nonlinear period of motion: dragon's head progresses through about 70% of a (nonlinear) period in 90 steps (about 129 steps per complete period), so the chosen timestep resolves the motion (ODE) reasonably well.

The simulation in this experiment was offline; reduced internal forces were computed using the direct formulas from Equation 2.23, as in [KLM01]. Such evaluation is currently the only practical approach to evaluate reduced forces for $r > 50$ on our hardware; computing the reduced cubic polynomial coefficients would in such cases result in too large memory footprints and evaluation times slower than the direct approach (appropriately sparsening the polynomials is a natural area of future research). Figures 2.18 and 2.20 show a sequence of animation poses computed under the different simulation methods, and Figures 2.19 and 2.21 give the deformation of one particular simulation mesh vertex, namely the vertex where the force load is applied.

Analysis of the results: Unreduced simulations exhibit very rich nonlinear dynamics which is difficult to capture with a subspace basis of limited dimensionality. The $r = 230$ and $r = 60$ bases perform reasonably well. The similarity of results for $r = 230$ and $r = 60$ suggests that there is a lot of statistical redundancy between the different derivatives: a 60-dimensional mass-PCA subspace already captures much of the "essence" of the full derivative 230-dimensional space.

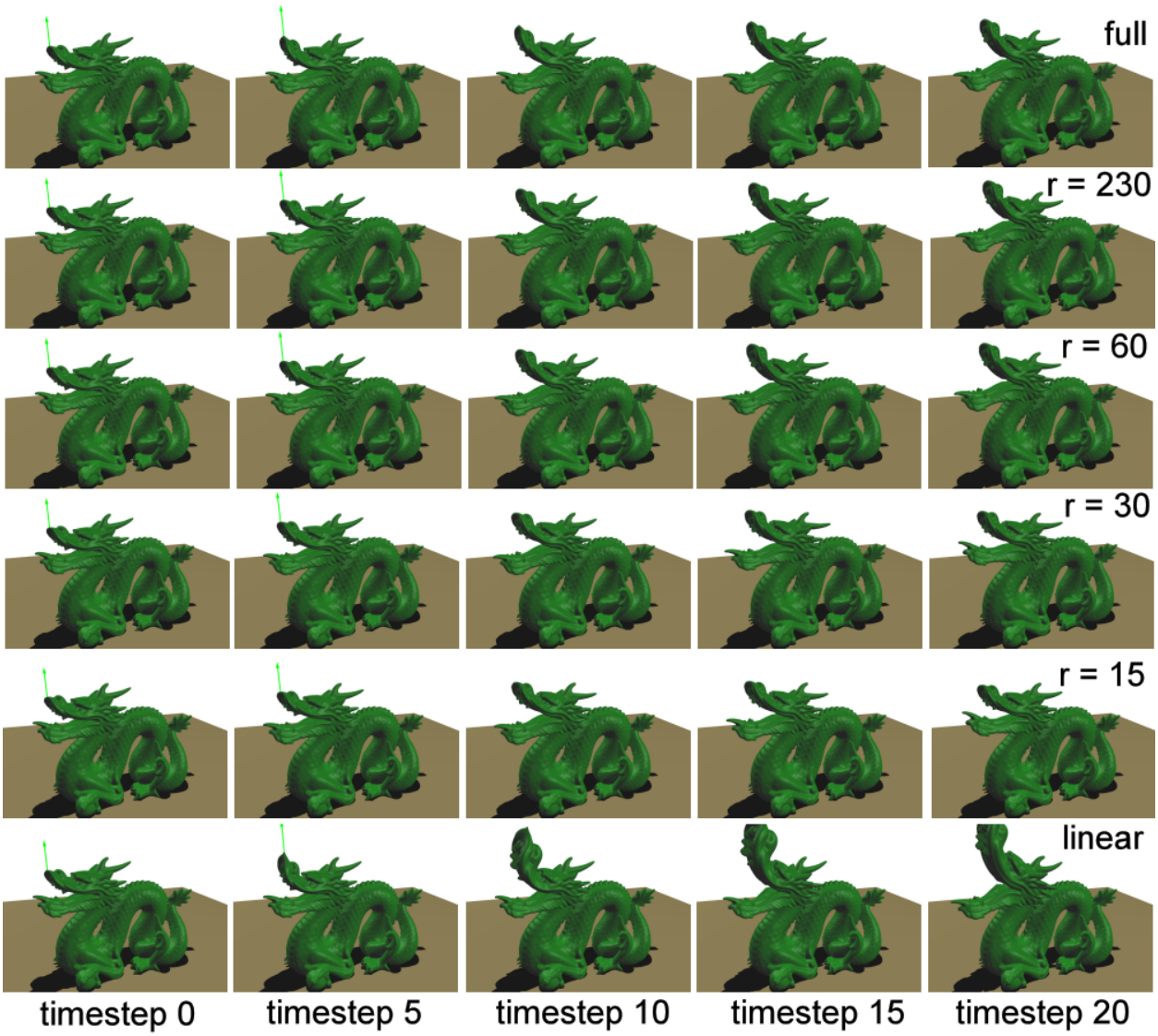


Figure 2.18: **Dragon "head" force load:** Dragon dynamic deformations, originating from a short vertical force on a central vertex at the tip of dragon's head. Each row gives deformations under one method, with columns corresponding to timesteps $0, 5, 10, 15, 20$, with a timestep of $1/300s$ (see Figure 2.19). All reduced simulations use $k = 20$.

Linear modes are essentially standing waves, and their spatial frequency typically increases with the mode index. The number of retained linear modes k therefore controls the range of spatial frequencies simulated, i.e., it controls how much local response the basis will be able to simulate. The parameter r controls to what extent the nonlinear correction to the first k linear modes will be available in the basis (lower modes are given priority through appropriate weighting, as described in Section 2.8.3). If r is only marginally larger than k, many derivatives (and also linear modes with

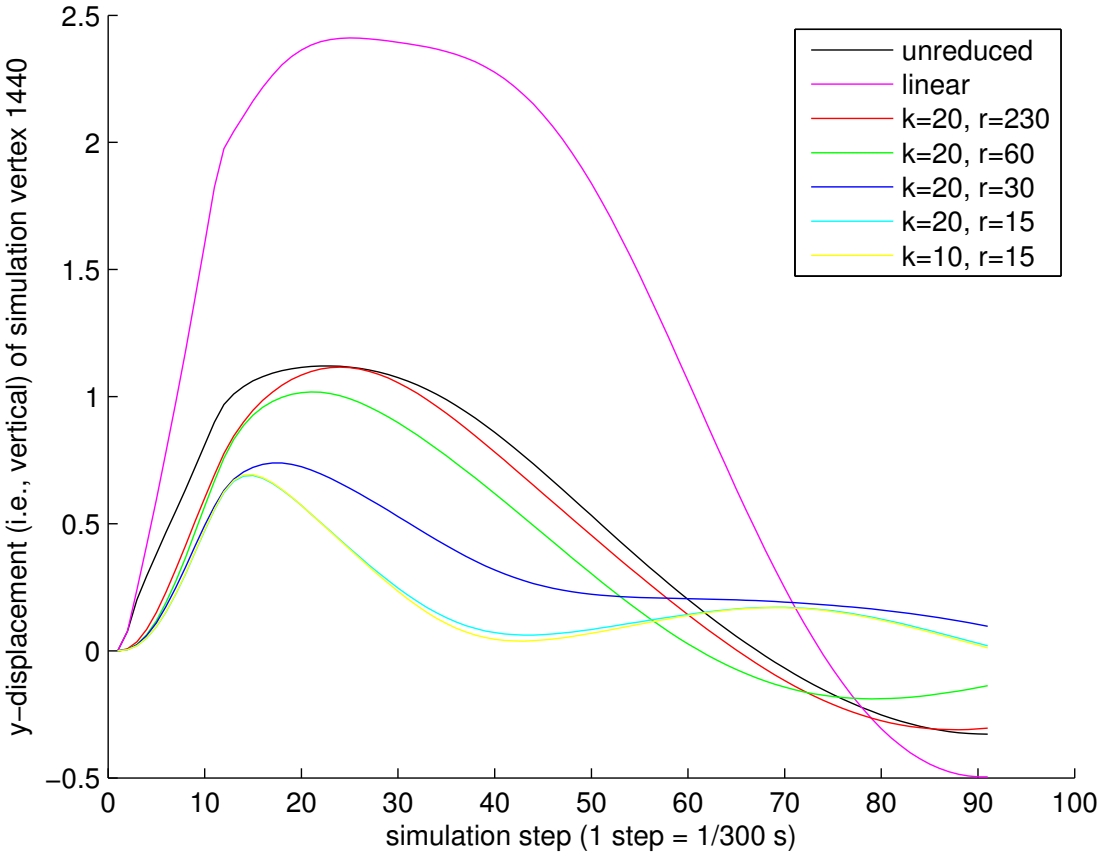


Figure 2.19: **Trajectory of a selected vertex:** Vertical displacement of simulation vertex where the force load was applied in the "head" sequence (Figure 2.18).

high indices) will not be included in the basis. Therefore only small deformation of these modes can be simulated. The bigger the r, the more complete the correction; if all $k(k+1)/2$ modes are added, one obtains a nonlinear reduced system capable of simulating moderate deformations away from the rest pose, up to the spatial frequency permitted by the retained k linear modes. The $r = 60$ curve suggests that a value of $r = 3k$ might suffice in many cases, i.e., it is not necessary to use the complete basis of dimensionality $r = k + k(k+1)/2$. Both Figures 2.18 and 2.20 plot the trajectory of the same vertex on which the force load was applied; this explains why the reduced simulation curves deviate from the unreduced curve already from the start: a very localized deformation occurs in the immediate vicinity of the pulled vertex, which is not captured in the reduced bases.

Artificial stiffening refers to the (erroneous) stiffening of the structure due to the simulation in a subspace (we also call it *locking*). Note, that "locking" is not to be understood as instantaneous

stiffening; our stiffening happens over finite time intervals, albeit possibly with a steep rate. Artificial stiffening is important for understanding subspace simulations. It explains, for example, why the subspace simulations tend to exhibit smaller extreme deformations than the unreduced simulations.

Artificial stiffening happens whenever the deformation into which the system wants to deform from the current configuration, given current deformation velocity and external forces, is not available in the basis. While in such cases an unreduced simulation would deform naturally by modifying the deformations of all vertices appropriately, a reduced simulation is forced to stay within a subspace, and therefore has to compromise by deforming into some less natural shape. Consequently, the strain energy grows much more quickly for the same amount of visual deformation, resulting in smaller deformation changes for the same increase in strain energy. In the absence of external forces the ODE's total system energy is invariant over time, regardless of whether the simulation is reduced or unreduced. Therefore, for the same initial kinetic energy, the extreme deformations tend to be smaller with reduced simulations, even though the extreme strain energy is the same. Artificial stiffening happens at the level of the ODE and is not specific to any particular numerical integration scheme. It represents the error due to simulation in a subspace. Some artificial stiffness is always present with reduced simulations, because unreduced solutions to the ODE never lie exactly in some low-dimensional space. Numerical solutions to the ODE introduce additional errors (e.g., numerical damping).

For example, artificial stiffening occurs if one performs a nonlinear StVK simulation in a basis consisting only of linear modes: the structure stiffens already for small deformations, since the derivative shapes are not available. It deforms very little beyond small deformations, even if larger force loads are applied. Another example is a structure that has already deformed to a moderate deformation within, say, the modal derivative basis, but the subsequent "natural" deformation deltas are no longer available in the basis (because the basis was computed around the origin). In practice, with a modal derivative basis, stiffening is not noticeable for small to moderate deformations (of course, this depends on k and r). In most cases, when stiffening occurs, it does not lead to visually implausible deformations but rather the system "locks" and can no longer continue to larger deformations. Of course, if force loads are removed, the system returns back to the rest pose.

Artificial stiffening due to the lack of a derivative in the basis can be observed with the "back" sequence (Figure 2.20, page 90, $r = 15, 30$). The sideways deflection linear mode is available in the basis, and the dragon does deflect, but very little (see right-most columns of Figure 2.20). In the $r = 60$ "back" sequence, deflection is larger as the derivative now is available in the basis. The $r = 230$ basis in addition enables the back of the dragon to curve sideways more interestingly (see third column of Figure 2.20). This is akin to simulating a cantilever beam using just the first deflection linear mode and its derivative, versus also including higher modes and derivatives.

Similarly, artificial stiffening is visible, say, in the $r = 15$ sequence of the "head" force load (Figure 2.18). The linear mode that tilts the head's pitch (almost rigidly) up and down is linear mode number 2. The linear mode that opens and closes the mouth is linear mode 5. These two modes are key shapes for capturing the deformations resulting from the "head" force load. With

$r = 15$, not all of the corresponding derivatives are in the basis, so we get artificial stiffening as the upper part of the mouth is pulled up (see right-most columns of Figure 2.18). In addition, in the unreduced simulation, the mouth not only opens, but the upper part of the mouth also bends (similar to a cantilever beam), as the external force is applied at the end of the mouth. The linear mode that would support such mouth-beam-bending does not appear within $k = 20$. This partially explains why the $k = 20, r = 230$ curve does not match the unreduced curve to a greater degree.

2.14 Real-time interactive demo

We made several of our pre-processed models available on the web, together with a demo executable where the user can interact with the models in real-time (by deforming the models with the mouse). We already performed all the precomputation (model reduction) for the provided models. We provide all the necessary data for runtime simulation.
Our demos are at: `http://graphics.cs.cmu.edu/projects/stvk/index.html`
Note that URLs are rarely permanent due to system maintenance issues and website migrations. Also, we cannot promise to keep the demos up-to-date if hardware, operating systems, etc., change in the future.

2.15 Discussion and future work

The deformable models presented in this thesis have a limited number of deformable degrees of freedom, and are as such not fully general deformable models. Obviously, the choice of the deformation subspace impacts the quality of the simulation. The algorithms in this thesis are designed to optimize the selection of the subspace to be as general as possible, without making any assumptions on the particular external forces imposed to the model at runtime. This is a difficult problem because there is no clear definition of "typical nonlinear deformations". The automatic basis selection approach that we developed selects a basis such that "natural" *global* large deformations of the model can be simulated. Global deformations are defined to be deformations with global support: (almost) every vertex of the mesh is deformed to some new position. Typically, the number of our global modes is too small to represent deformations involving high spatial frequencies. Local deformations therefore cannot be simulated without excessive computational effort: it would be necessary to add appropriate localized deformations into the basis, and doing so for all localities on the model results in bases with so many elements (too high r) that the model is no longer interactive.

Another limitation is the $O(r^4)$ run-time complexity of evaluating the StVK reduced internal forces and stiffness matrices. For small values of r this is not a major issue: we have demonstrated real-time performance for values as large as $r = 30$, and the largest value of r ever used for a simulation in our experiments was $r = 60$. However, the quartic $O(r^4)$ complexity prevents the method from reaching even higher values of r. If the motion basis is spatially sparse (not the case with motion bases in this thesis), then a lot of the terms in the cubic polynomials become zero. A spatially adaptive method could therefore avoid the $O(r^4)$ complexity; the challenge is how to

(simultaneously) construct a multi-resolution hierarchy of domains adapted to the object geometry and material distribution, together with appropriate basis functions on each domain.

During contact, the boundary conditions of the model effectively change in the contact region. This makes the precomputed motion basis less optimal, since this basis was determined under the contact-free boundary conditions. Deformations in our thesis are large and self-collisions can occur in extreme poses. Self-collisions were not a focus of this thesis, but could be addressed in the future, for example by augmenting the Bounded Deformation Tree [JP04] method to detect self-collisions efficiently. During self-contact, basis refinement may be required due to the changed boundary conditions. Precomputation "bakes in" the topology of the mesh, and as such, interactive cutting is not possible. The StVK material also does not work well under extreme element compression (see comments on element inversion in the Related work section). All of these issues are topics of future work.

A modal derivative basis for unconstrained models (Section 2.8.7) can accommodate multi-body dynamics where objects are in free flight. In the equations of motion for such a system, the rigid part is coupled with the deformations, accounting for effects such as the inertia tensor changing under deformation, or the elongation of the object under centrifugal forces arising from large angular velocities [Sha05]. For low velocities, however, one can use the approximation of integrating the rigid body motion separately from the deformations. We use this approximation for all the unanchored large deformation simulations in this thesis. Under this simplified model, the external forces excite the deformations, plus they also applied to the rigid body motion part of the simulation. Each of the two parts is then integrated separately.

The reduced forces only take the form of a cubic polynomial for linear materials. While dimensional model reduction is possible with arbitrary materials, the reduced forces are in general not easy to evaluate directly from the reduced coordinates. Instead, they must be constructed by first evaluating full unreduced forces, followed by a subspace projection. However, even in this general nonlinear case the reduction approach has the benefit of replacing the expensive solution to a large sparse linear system by that of a small dense $r \times r$ linear system.

For certain isolated extreme deformation poses, and for extremely low values of r (e.g. $r = 2$ for the bridge), the reduced internal force field can contain spurious stable equilibriums. This is a manifestation of the fact that the chosen value of r is simply too small to represent the problem. In our experience, this problem can always be solved by increasing r.

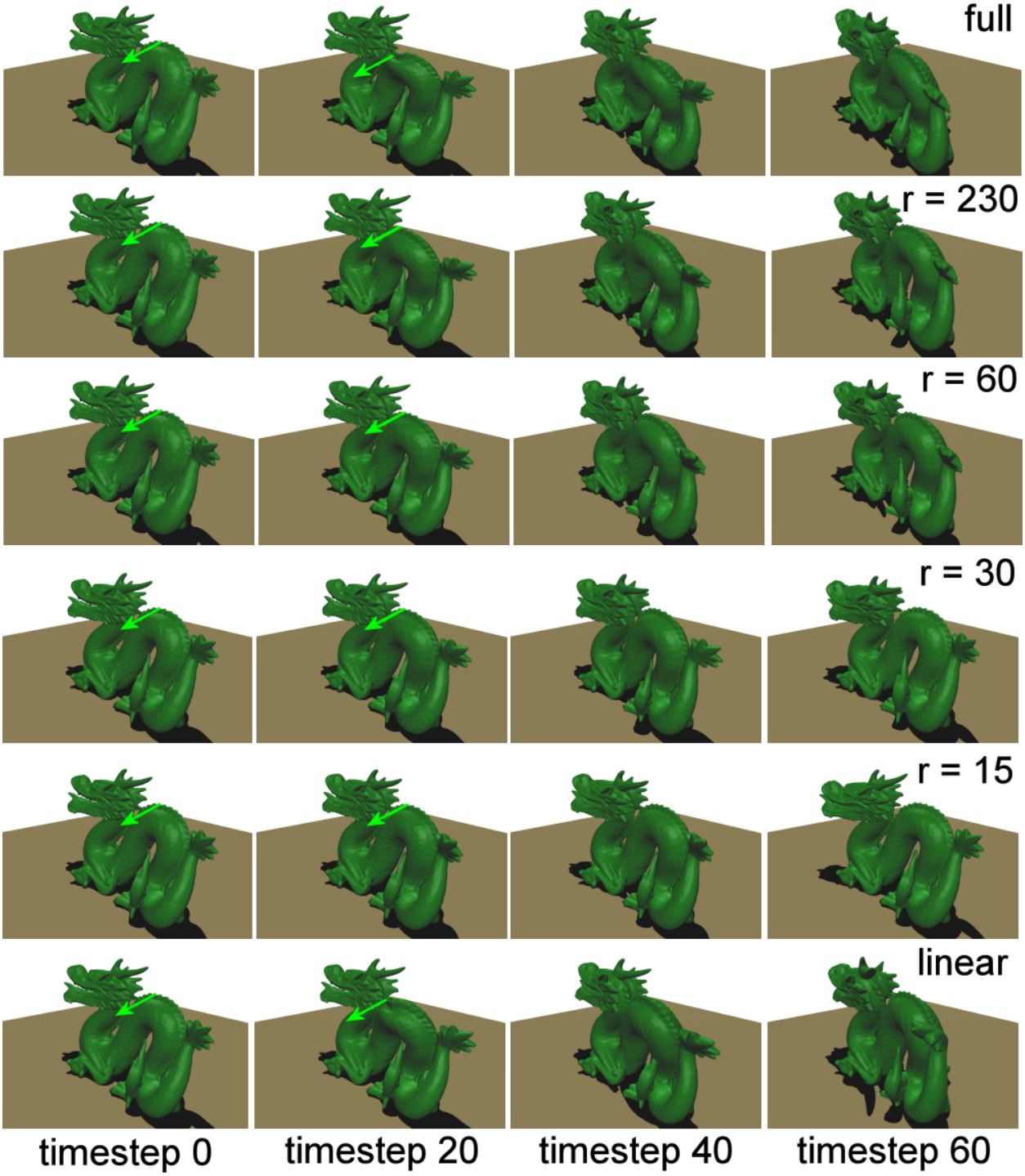


Figure 2.20: **Dragon "back" force load:** Dragon dynamic deformations, originating from a short force (horizontal, perpendicular to dragon's main axis) on a central vertex at the top of dragon's back. Each row gives deformation under one method, with columns corresponding to timesteps $0, 20, 40, 60$ (see Figure 2.21). All reduced simulations use $k = 20$.

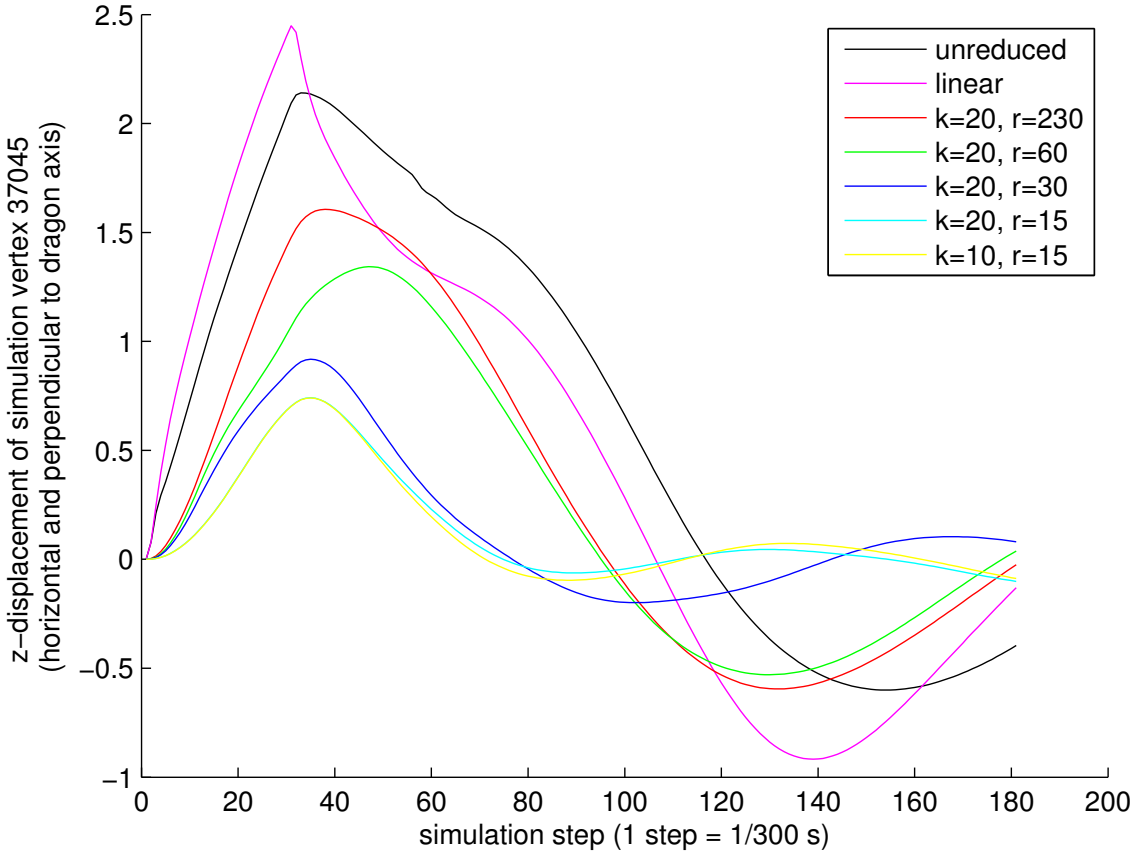


Figure 2.21: **Trajectory of a selected vertex:** Z-displacement (i.e., horizontal and perpendicular to main dragon axis) of simulation vertex where force load was applied in the "back" sequence (Figure 2.20).

Chapter 3

Time-critical distributed deformable contact

This chapter presents a novel CPU-based algorithm to simulate contact between *geometrically detailed* reduced deformable objects and *geometrically detailed* rigid objects. Three types of interaction are supported: rigid vs rigid, reduced-deformable vs rigid, and reduced-deformable vs reduced-deformable. Such simulations (especially the deformable simulations) are challenging because collision detection, contact force computation and deformable object dynamics are all time-consuming operations. Consequently, maintaining haptic update rates is difficult. To the best of our knowledge, this thesis presents the first deformable haptic simulations with geometric detail on the order of magnitude depicted in our examples. The reduced deformable object simulations are achieved using the algorithms presented in the previous chapter of this thesis.

Researchers at the Boeing Company have invented the Voxmap-PointShell method (VPS), which is a pioneering six-degree-of-freedom haptic rendering algorithm capable of rendering distributed contact between a rigid object and a detailed rigid environment. The contact is *distributed* (also sometimes called *multi-point* contact, see Figure 3.1): there can be several simultaneous contact sites each spatially distributed over a (typically non-zero) surface area. The contact surface areas and tractions can vary from one site to the next. Distributed contact represents an important component of interactive physically based virtual environments. It is the most general kind of contact between two (detailed) objects. Contrast this with simulations where two objects are only allowed to touch simultaneously at a single point, or even simpler simulations where a single object can only be touched by a single point, or the user can pull on selected vertices of the model.

VPS was introduced in Boeing's SIGGRAPH 1999 paper [MPT99], and subsequently improved in [WM03] and [MPT06]. Typical environments simulated consisted of the interior of a commercial airplane, with hoses, linkages, tubes, wires, bolts, and different other mechanical parts. The mechanical parts are often in close proximity to each other, or even fit tightly to each other, introducing a lot of opportunity for distributed contact. Haptic simulations are useful for virtual prototyping: airplane designers need to check if a certain mechanical part can be positioned into/removed from its designated location, so that design flaws can be detected without actually

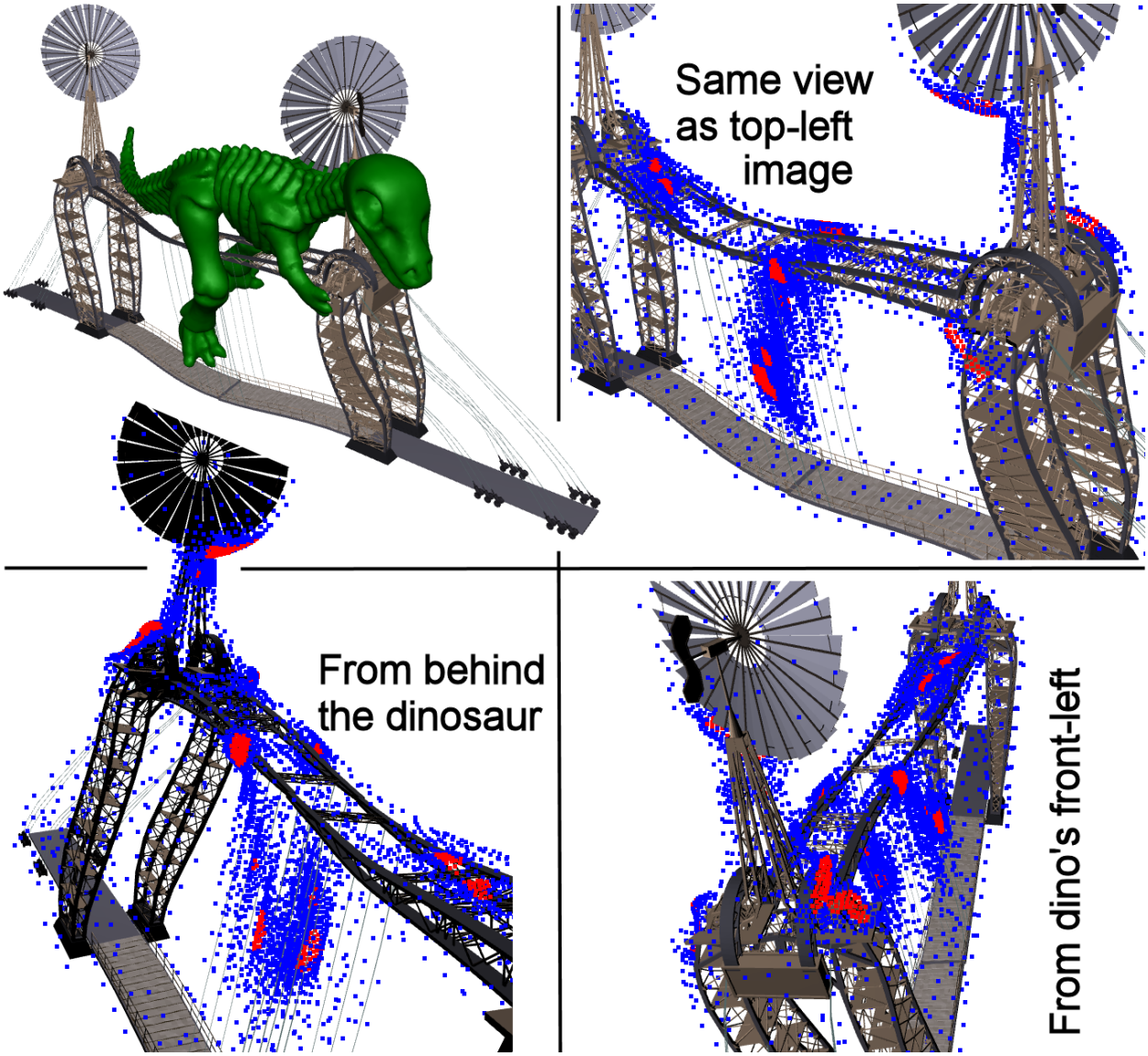


Figure 3.1: **Distributed deformable contact:** Top Left: contact configuration between a nonlinear reduced deformable bridge and a rigid dinosaur, with 12 separate contact sites. Other images: contact sites viewed from three different camera angles. A total of 5200 points were traversed, resulting in 426 points (in red) in contact. Points traversed but not in contact are shown in blue. Note: temporal coherence was disabled for this figure.

manufacturing the structure. In this way, one can test whether a fire detector can be removed from an aircraft engine, such as when the detector needs servicing. Another example presented in the literature is manipulation of a small valve in close proximity to aircraft geometry such as beams, wires, and tubes [WM03].

Virtual prototyping can be approached with conventional path-planning methods. Haptic simulations, however, have the promise of being simpler and more reliable. In a haptic simulation, the manipulated object is virtually attached to the haptic device's manipulandum, and the human operator can then attempt to insert or remove the mechanical part, while the simulation ensures that the penetration constraints are not violated. Any contact forces are haptically rendered to the operator, which helps the operator find the desired path. Such *haptic path planning* exploits human path planning intuition to potentially solve difficult planning problems faster than using fully automated path planners. The path from the haptic simulation could serve as an initial guess to an offline fully automated path planner [BSA00].

Existing virtual assembly simulations are, however, limited to rigid objects and environments (for geometrically complex models). In a deformable object simulation, a significant amount of computation must be devoted to deformations, which makes it difficult to maintain high haptic update rates. Simulating deformable geometry is important: airplane geometry is often rigid, but hoses, wires, tubes, etc., are deformable. If a rigid simulation is used to check for mechanical part removability, it might be unnecessarily restrictive and report that a part cannot be (dis)assembled, when in reality it can be. For example, wires and hoses can be bent to let the object pass through a narrow passage (see Figure 1.4 on page 29).

3.1 Background: The Voxmap-PointShell (VPS) method

VPS is a haptic rendering method that can simulate six-degrees-of-freedom distributed contact between a rigid object and a rigid environment. The rigid object is manipulated by the haptic device. The user can position the object in the environment interactively, and feel the forces and torques as the object collides with the environment.

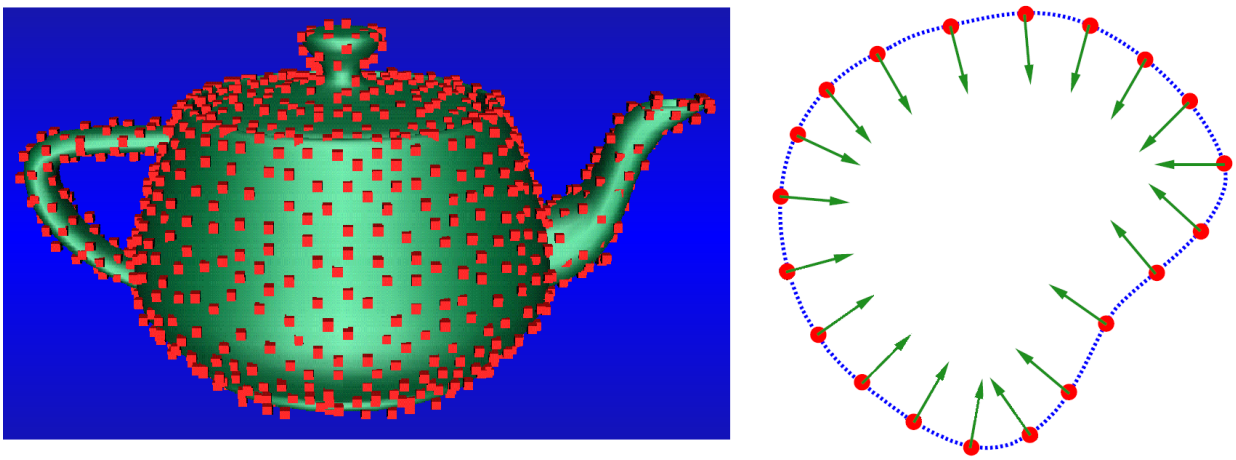


Figure 3.2: **VPS pointshell:** Left: the pointshell points for the teapot model. Right: a schematic representation of the pointshell, with inward normals. Both images are from [MPT99].

VPS uses a different representation for the rigid object and the rigid environment, and is as such an asymmetric approach. The rigid object is modeled as a pointshell: a collection of points positioned near-regularly on the surface of the object, similar to surfels in point-based rendering (see Figure 3.2). Each point also caches its inward normal, which is static since the object is rigid. The environment is modeled as a *voxmap*, a compressed data structure computed by voxelizing polygon geometry into small voxels (see Figure 3.3). Each voxel is assigned a 2-bit value:

- 0: *free-space*: voxel lies in the exterior empty space and is not a proximity voxel of case 3
- 1: *interior*: voxel lies inside some environment object
- 2: *surface*: voxel intersects surface of some environment object
- 3: *surface proximity*: voxel is immediate neighbor to some surface voxel, and is non-interior

In this sense, the voxmap can be seen as a 2-bit implicit function, designed specifically for the purposes of haptic rendering. The improved VPS approach of [MPT06] allocates more bits per voxels so that they can cache distance-to-contact information with each voxel. Interesting environments require high-resolution voxmaps, which can easily exceed the size of a computer's main memory. To save space, the voxmap is compressed by an octree: if all voxels in a certain subtree of the octree contain the same value v, the subtree can be pruned and replaced by a leaf node storing v.

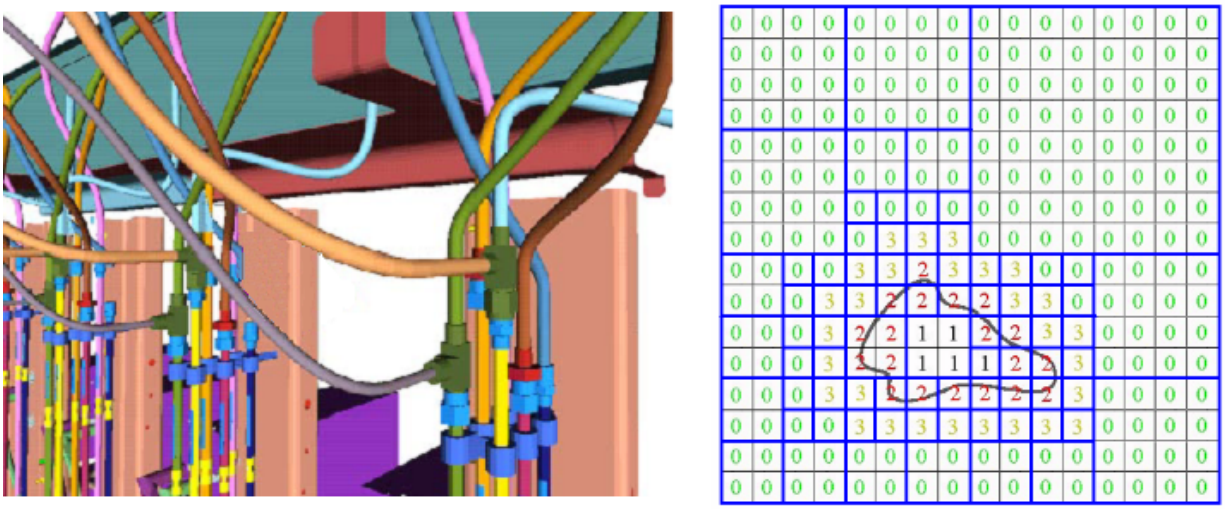


Figure 3.3: **VPS voxmap:** Left: a VPS environment (a scene from the interior of an airplane, serving as source data for voxmap). Right: a schematic illustration of the voxmap, also showing the voxmap octree. Left image is from [MPT99]. Right image is an enhanced version of an image from [MPT99].

The voxmap and pointshell are precomputed, so that they can be used efficiently for fast runtime haptic simulation. At runtime, at every haptic cycle, every pointshell point is queried against

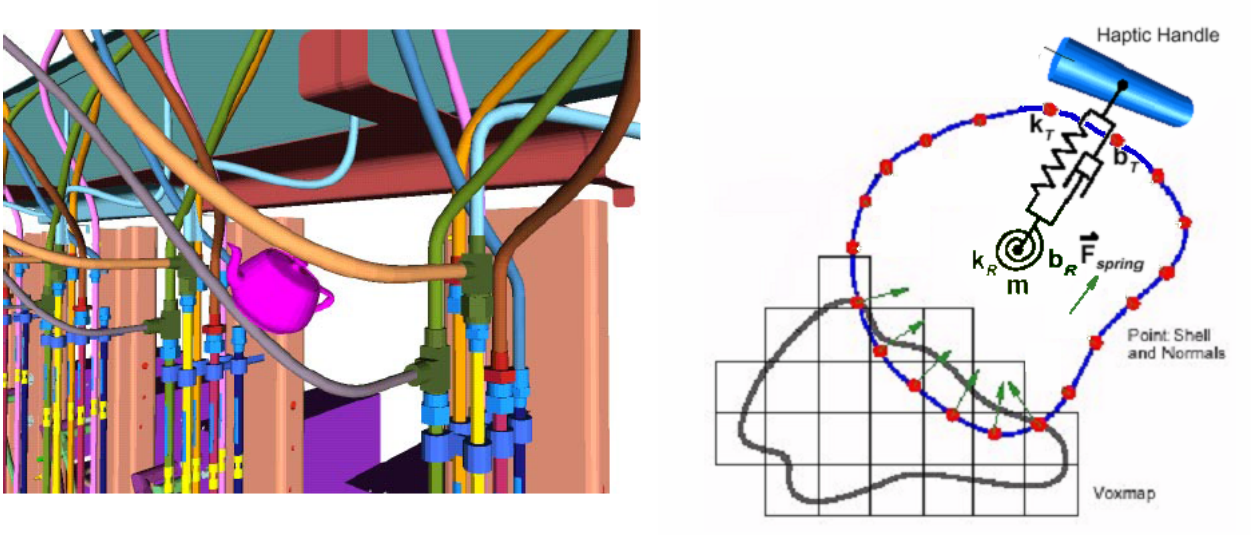


Figure 3.4: **VPS in action:** Left: the user manipulates the teapot in the virtual environment. Right: the virtual coupling between the haptic device and the manipulated simulation object in contact. Both images are from [MPT99].

the voxmap (see Figure 3.4). No action is taken if the point lies in the exterior empty space. If the point occupies a surface voxel, a penalty force is assigned to the point, pushing it away from the object. The direction of the force equals the point's cached static inward normal, and the magnitude is determined from the amount of penetration. The algorithm is designed to prevent points from entering into the interior region of the model. Preventing such penetration is not trivial to achieve, and VPS takes special care to prevent it as much as possible. Proximity voxels are treated just like free-space voxels, with the addition that *breaking pulses* (short impulsive contact forces) [SV97] are applied to high-velocity points in contact (if the velocity is large enough and directed toward contact). Such breaking pulses prevent the user from propelling the object toward the environment, overpowering the surface voxels' penalty force layer to penetrate into the environment's interior. While these kind of situations are of course not typical of manipulations with careful intent, it is still important for the method to respond appropriately if they do occur.

Pointshell accuracy and virtual coupling stability have been improved in [RPP+01]. The Boeing follow-up paper of [WM03] replaced the original VPS mass-spring-damper for a quasi-static spring, introducing *static virtual coupling* (which is discussed extensively below). In [MPT06], the authors introduce the additional constraint that the pointshell object is only allowed to move with a certain maximum velocity (the *MaxTravel* constraint), which further prevents such penetrations. Note that both linear velocity and angular velocity need to be limited. Another improvement of [MPT06] is wider voxmaps (4-bit), to accommodate distance-to-nearest-contact information for free space voxels. Since the haptic device is controlled by a human, there exists an upper bound on the maximum speed of a pointshell point, and therefore points located in a voxel sufficiently far from contact need not be checked for several haptic cycles (temporal coherence). Also,

the pointshell is traversed on two levels: a coarse level pointshell is tested against a coarse level voxmap, followed by finer tests if necessary. Unlike the methods presented in this thesis, the hierarchy of [MPT06] only consists of two levels, the simulation is rigid, and graceful degradation of contact was not addressed.

3.1.1 Virtual coupling

Six-DOF haptic rendering is not simply a matter of computing contact forces and torques at high rates. If the contact forces and torques are rendered directly to the user ("direct rendering"), the simulation will suffer from stability problems. For example, the VPS contact force magnitude jumps discontinuously to zero when a point leaves a surface voxel and enters a proximity voxel (say, under motion tangential to the contact normal). Direct rendering is also problematic when several points enter contact simultaneously (which is a common occurrence). Each point adds to the overall stiffness of the system, and together they can exceed the maximum stiffness of the haptic device. Maximum renderable stiffness is an important hardware parameter of each particular (impedance) haptic device. If the simulation produces contact forces that are stiffer than maximum device stiffness, and if these forces are directly rendered to the device, the device starts to buzz and eventually goes unstable. For example, we observed the maximum renderable stiffness of our PHANToM Premium 1.5A 6-DOF-output haptic device to be about $0.6\,\mathrm{N/mm}$.

To mitigate these problems, the penalty forces and torques are not rendered directly to the user. Instead, it is customary to separate the simulation position of the haptic object (the *simulation object position*) from the position imposed by the haptic manipulandum (the *haptic manipulandum position*), and connect the two with a spring (*virtual coupling* [CMJ95, AH98a]) (see Figure 3.5). The penalty spring tries to align the simulation position and orientation of the haptic object to the manipulandum. Virtual coupling sometimes also includes damping, to enforce equal velocities in addition to positions. There are actually two separate 3-DOF virtual coupling springs: one for the linear translations and one for rotations. It is possible to prove that the stiffness of virtual coupling forces presents an upper bound on the stiffness ever rendered to the haptic device, no matter how many points might be in contact simultaneously (assuming continuous time control, i.e., infinite haptic update rates). We give the proof in Section 4.7. The displacement-force relationship of virtual coupling is designed to be linear only up to a certain maximum force value, after which the force is held constant (saturates) at some maximum value, even if the user stretches the virtual coupling spring even further. Such saturation prevents the user from pulling the manipulated object into the interior of the other object. For more details, see Section 4.9.

We consider two kinds of virtual coupling: dynamic virtual coupling (which uses mass-spring-damper) and static virtual coupling (which uses spring only).

Dynamic virtual coupling

In dynamic virtual coupling [CMJ95, AH98a], the simulation object is assigned a non-zero mass, and virtual coupling is a mass-spring-damper. At the beginning of every haptic cycle, the simula-

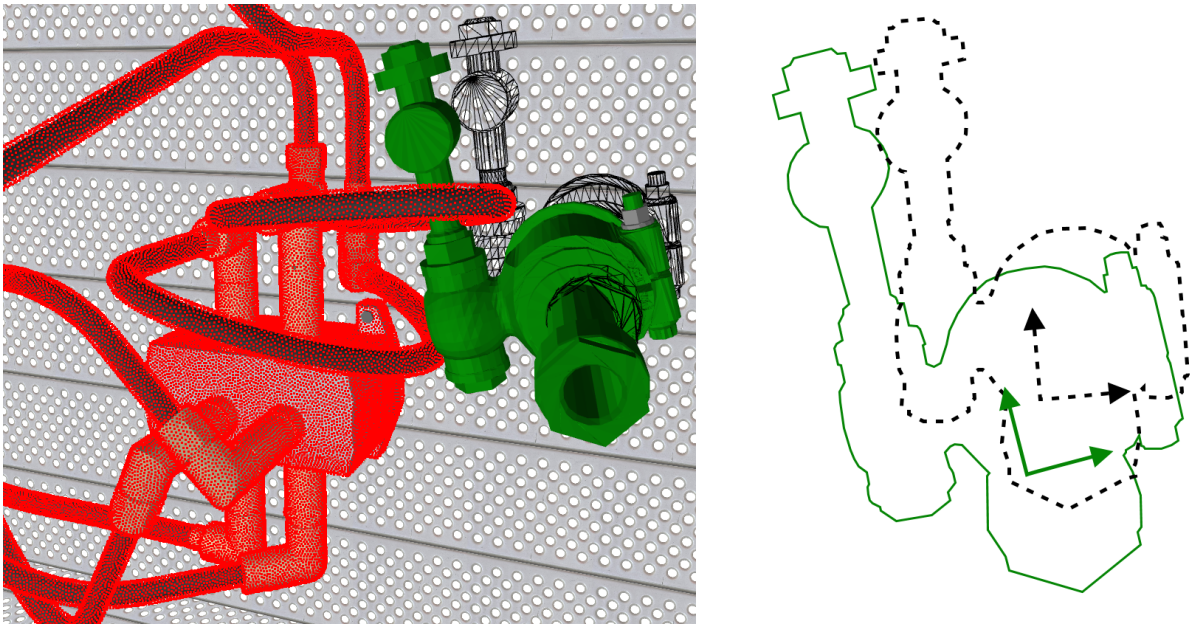


Figure 3.5: **Virtual coupling:** Left: the green mechanical component (distance field haptic object) and the deformable hose scene (pointshell, with points shown in red). Wireframe shows the manipulandum position, solid green color shows the simulation object position. Right: a schematic representation of the manipulandum position (stippled black line) and simulation position (solid green line), together with the two coordinate systems which the virtual coupling tries to align.

tion reads the current haptic manipulandum position and orientation. This position and orientation are then set to be the current rest position and orientation of the two 3-DOF virtual coupling springs. Since the simulation object position will (almost) never match the haptic manipulandum position (even if they matched during the previous haptic cycle, the shaking of the user's hand will cause a slight mismatch at the beginning of the next cycle), there will be some non-zero virtual coupling force $F_{\mathrm{VC,spring}}$ and torque $T_{\mathrm{VC,spring}}$. The difference in velocities (and angular velocities) of the haptic manipulandum and simulation object is penalized by damping forces (and torques) $F_{\mathrm{VC,damping}}$ and $T_{\mathrm{VC,damping}}$. The total virtual coupling force and torque are then $F_{\mathrm{VC,spring}} + F_{\mathrm{VC,damping}}$ and $T_{\mathrm{VC,spring}} + T_{\mathrm{VC,damping}}$, respectively. Next, collision detection and contact computation are invoked with respect to the *simulation* object position. The computed contact forces and torques are added to the virtual coupling forces and torques, which gives the net forces and torques on the simulation object. Next, the position and orientation of the simulation object is updated using the equations of motion of a rigid body. These equations take into account the simulation object's mass and inertia tensor.

The original VPS method of [MPT99] uses dynamic virtual coupling. The inertia tensor was modeled as a diagonal matrix to simplify the simulation. The rigid body motion was integrated using the explicit Euler integrator. We found that simulation stability can be slightly improved if we subdivide the haptic timestep into several Euler sub-timesteps. During these sub-timesteps, the

haptic manipulandum position and orientation are assumed to stay constant (or could be extrapolated from previous positions, e.g., using a Kalman filter). The Euler integrator updates the simulation object position and orientation after every sub-timestep (which causes the virtual coupling forces and torques to change). The contact forces and torques are kept constant (again, extrapolation would be possible), as they are expensive to compute (they require traversing the pointshell). For this reason, the stability improvements due to sub-timestepping are limited. However, we speculate that if manipulandum position and orientation could be sampled continuously (infinitely fast), and contact forces and torques also sampled continuously (with zero computation cost), and if one could (say analytically) solve the resulting ODE for the simulation object rigid body motion with zero latency, that this would give very stable force and torque signals. Such (hypothetical) simulations are called simulations with *continuous time control* [CGSS93]. In essence, our simulations are discrete approximations to this continuous limit. Of course, the limit can never be achieved in practice. It is useful from a theoretical point of view, as stability properties can sometimes only be proven under continuous time control, with discrete-sample time control converging to the limit under progressively higher update rates.

Virtual coupling damping requires measuring manipulandum velocities. The rotational damping of [MPT99] used only the angular velocity of the simulation object, not the difference in angular velocities of the simulation object and haptic manipulandum. The API of the PHANToM force feedback device does not (currently) support measuring manipulandum's angular velocity, only translational velocity. Even for translational velocities, accurate estimates are inherently difficult to obtain due to limited encoder resolution. The need to measure velocities presents one weakness of the dynamic virtual coupling model. In our experiments with the PHANToM device, the velocity estimates were often under-resolved (noisy) to the point that they would degrade our haptic signal. If the damping parameters are set too high, the noise in the measured velocities can amplify into a haptic instability. Low damping, in turn, causes instabilities during contact, when more damping would be needed to make the contact more passive. In practice, we found it challenging to choose good virtual coupling damping parameters. Furthermore, one also has to select good values for virtual coupling force stiffness, and the virtual object mass. Overall, we spent a significant amount of time tuning virtual coupling mass, damping and stiffness parameters, and the results were still not satisfactory in some cases. For example, we were sometimes not able to render contact forces and torques as stiff as permitted by our hardware. In such cases, we were always able to make the dynamic virtual coupling simulation stable by decreasing stiffnesses, at the cost of making the contact less stiff (and therefore less realistic).

With badly tuned parameters it is not difficult to run into residual instabilities when the simulation object is located in free space. One approach to avoid the residual instabilities is to simply render zero forces and torques if the simulation object is not in contact with the environment. However, this is not a perfect solution. For example, consider a case where a square peg is inserted into a (long) vertical square shaft. The contact forces in this case provide no resistance to the upward/downward motion of the manipulandum: along this axis, the simulation behaves as if in free space. However, the virtual forces and torques *will* be rendered in this case (together with any instabilities), because the object is in contact. There are several other such "degenerate" contact

situations, such as a cylindrical peg in a cylindrical hole, ball in a (hemispherical) socket, or a box in between two planes.

Static virtual coupling

Static virtual coupling was introduced in [WM03], where they replaced the mass-spring-damper with a quasi-static spring. Static virtual coupling simplifies the simulation, since no mass or damping parameters are needed; what is needed is only one scalar parameter for translational virtual coupling stiffness, and one scalar parameter for rotational stiffness. Also, it is no longer necessary to measure any manipulandum velocities. This paper also modified the virtual coupling displacement-force relationship for large spring extensions. The original virtual coupling used a linear displacement-force relationship, followed by a constant saturated maximum force. The new relationship also uses an initial linear region, but then exponentially saturates the force to a maximum force value, making the displacement-force relationship continuously differentiable. The exponential relationship can be replaced for a computationally simpler cubic spline [OL05]. Our experiments indicate, however, that the difference in execution speed is very small. The cost of evaluating one exponential function value per haptic cycle, or evaluating one cubic polynomial per haptic cycle, are negligible compared to the costs of collision detection, contact computation and deformable object simulation.

In this thesis, we adopt *static* virtual coupling of [WM03] since it needs a relatively small number of parameters, and extend it by introducing *static damping*. According to our experiments, static virtual coupling can simulate very stiff environments. We were able to maintain stability while increasing stiffness all the way up to hardware stiffness limits of the PHANToM force-feedback device (e.g., 0.6N/mm). However, the static model is not forgiving to contact force/torque discontinuities, and in the case of voxmap forces of [WM03], those discontinuities can be felt clearly. This motivated us to use a distance field instead of a voxmap, where contact forces and torques are continuous with respect to simulation object position and orientation. *We were able to feel a clear improvement between distance field contact forces and voxmap-based contact forces.* Another improvement presented in this thesis is the inclusion of large rotation terms in the static equilibrium equations (which are not included in formulas in [WM03]). We also demonstrate how to handle virtual coupling saturation in the presence of rank-deficient virtual coupling force and torque gradients of static virtual coupling. These improvements are presented in Chapter 4.

3.2 Overview of our approach

As described previously, this thesis addresses distributed contact between 3D objects, i.e., contact with potentially several simultaneous contact sites, each distributed over a non-zero surface area. We use a penalty-based contact model similar to VPS: contact force direction is obtained from the point's normal, and the magnitude is proportional to the amount of penetration. We obtain this penetration depth from a precomputed floating-point signed distance field instead of a

voxmap. The contact model is described in Section 3.5. We deform our pointshells using the reduced nonlinear deformable object algorithm presented in the first part of this thesis. We give the details in Section 3.6. Pointshell normals change under deformation, and we update them using the approximation described in Section 3.6.2. A rigid distance field and a deformable pointshell support haptic contact between a geometrically complex rigid and geometrically complex reduced deformable model for the first time. In addition, we present a novel extension that permits approximate evaluation of deforming distance fields for reduced deformable models, which enables haptic rendering of two geometrically complex reduced deformable objects in contact. Deformed distance fields are presented in Section 3.12.

To support haptic display of geometrically detailed models, e.g., involving a million pointshell points, we resolve distributed contact progressively. We organize the pointshell into a *nested* multiresolution hierarchy, by sampling points (and normals) from an appropriately defined *smooth manifold surface*. Our sampling is near-uniform which provides economic pointshells and improves haptic stability. We use a precomputed sphere-tree hierarchy to bound the pointshell points, *with sphere centers located at (non-leaf) pointshell points*. For deformable models, this is a point-based variation of the Bounded Deformation Tree (BD-Tree) [JP04]. Such a hierarchy permits large pointshells, bigger than what could be examined point by point during one haptic cycle: if a point on some coarse hierarchical level is sufficiently far from contact, none of the points in the subtree can be in contact, and then those points need not be checked. In contrast, if a large part of the pointshell is in contact or close to contact, few subtrees will be culled and a large part of the pointshell will need to be examined point by point. We address this problem by providing *graceful degradation of contact*: if there is not enough computation time to fully complete the tree traversal, the algorithm still returns a reasonable answer, with accuracy dependent on the contact configuration difficulty and the available processing power. Graceful degradation is achieved by traversing the nested hierarchy in a breadth-first order, rendering deeper and deeper tree levels until out of computation time. We use two separate activation thresholds to avoid abrupt changes in the rendered depth during consecutive haptic cycles. The simulation also uses temporal coherence to time-sample the individual points at rates depending on distance to contact, i.e., temporal adaptivity.

We display distributed contact (haptically) using a novel first-order quasi-dynamic virtual coupling method which uses both forces and torques, and their derivatives with respect to simulation object position and orientation. In haptics, higher update rates generally enable stable rendering of stiffer forces and torques; but reaching such high rates is challenging for models with detailed geometry. In our system, the deformations, collision detection and contact force computations run together in one loop at haptic rates (1000 Hz). We do not extrapolate from lower-rate signals. A multi-rate simulation using the contact gradients of Section 4.2 could exploit user's limited sensing bandwidth to further increase example complexity [OL06, OG07].

3.3 Related work: Multi-resolution rendering

The idea to render point clouds hierarchically is very natural, and has been previously often employed in various computer graphics applications. Our interruptible point-based contact-sampling approach is analogous to the QSplat algorithm from point-based rendering [RL00] which represented the point cloud hierarchically to enable level-of-detail control for *graphical* rendering. Our oriented pointshell can be seen as a surfel point cloud, the meshless geometric representation used in point-based rendering. A single-resolution contact algorithm for two mesh-free deformable point-sampled objects has been presented in [KMH+04]. The method computes the contact surface using the point-based surface projection operator, reaching interactive (but not haptic) rates: timesteps of 130-200 milliseconds were reported for two models of about 10,000 surfels each. Hierarchical point-contact handling was explored in [PPG04], where they demonstrate near-interactive (a few frames per second without precomputation) contact of point-based quasi-rigid (quasi-static and linear) deformable objects. This particular deformable model was designed for local deformations; global deformations were not modeled. Contact tractions are computed by approximately solving LCPs in each contact region, at progressive resolution levels, until desired accuracy is reached. In this thesis, we present a hierarchical contact rendering method designed for high-rate (haptic rendering) applications. We demonstrate how to maintain upper bounds on the amount of computation per haptic cycle, and how to minimize contact force artifacts when run-time constraints force a transition to a different contact resolution level.

Bounding volume hierarchies can be used for *interruptible* collision detection for rigid [Hub95] and embedded deformable models [MO06] by traversing the hierarchy in a breadth-first manner. Time-critical contact can be approximated by applying contact forces between contacting (sphere) primitives at the deepest level traversed in a particular frame. Unfortunately, such approaches can have trouble resolving close-proximity and conforming contact configurations in high-rate scenarios (as in haptic rendering): finer bounding volumes more closely approximate the object, and as such the effective contact surface (and therefore forces) depend on the level of detail rendered at any instant. In our work, the points at different resolution levels progressively sample the same well-defined contact surface, thereby making contact at different levels of detail more consistent.

Six-DOF haptic rendering of complex *rigid* objects can be achieved using a number of techniques, in addition to VPS, such as multi-resolution collision detection on convex decompositions, followed by a clustering of contacts [KOLM03], or incremental local contact search [GME+00]. Multi-resolution contact haptics has been demonstrated for polygonal geometry by appropriately simplifying the objects at different hierarchical levels, to preserve the haptic *sensation* to the user [OL03, OL06]. The authors suggest that for contact sites of large area, resolving the contact at a coarser resolution will not result in significant haptic sensation degradation; this has motivated our graceful degradation algorithm. These approaches, however, do not directly apply to point-sampled objects or deformable objects.

3.4 Related work: Haptic rendering of deformable objects

Haptic rendering has received increasing attention over the last decade [LD07], and significant challenges remain especially for rendering deformable systems. Early methods were developed for tissue simulations during virtual surgery [KcM00], and mostly used mass-spring deformable objects. Space-time adaptive and multi-rate methods have been successful for real-time deformable object simulation [AH98b, DDCB01], but time-critical distributed contact algorithms have received less attention (c.f. [MO06]). Discrete Green's functions (computed, say, under the BEM discretization) can be used for haptic rendering of linear elastostatic solid 3D deformable models [JP01, JP03]. The simulation is linear and as such accurate under small deformations. The approach can model static deformation response under externally applied forces and vertex position constraints. The simulation is driven by the user prescribing the position of a single vertex (or a few vertices) using the haptic manipulandum, and feeling the force as the model deforms to accommodate prescribed position of the vertex (or vertices). The static deformation response of the model is encoded in the columns of the Green's function matrix; roughly speaking, each column gives the deformation under the displacement of a certain vertex in one of the x, y, z directions. With some extra run-time computations, the technique can be extended to the case where positions of multiple vertices are prescribed simultaneously. Let $\mathcal{S}$ denote the time-varying set of vertices constrained during a particular simulation timestep. There is a separate set of Green's functions for every possible set $\mathcal{S}$ and, since the number of possible sets $\mathcal{S}$ grows exponentially with the cardinality of $\mathcal{S}$, it is not feasible to precompute these Green's functions for all possible sets $\mathcal{S}$. Instead, these updates are performed at runtime, by incrementally updating the Green's functions using the Sherman-Morrison-Woodbury formula. The approach is fast, compact, robust and well-suited for local deformations. One limitation is that for large contact areas with many vertices simultaneously in contact, the required updates become progressively expensive, or more complicated hierarchical GF strategies are needed. Also, the approach cannot simulate deformable dynamics and large deformations.

Related to the Green's function approach are the elastostatic FEM approaches of [BNC96, CDA99], employing techniques of static condensation and precomputed model displacements for unit forces applied on the object's boundary. These approaches use a different kind of discretization (FEM), and require a volumetric mesh. A simple linear correction scheme is suggested to extend the single-contact responses to contacts with multiple points. A recent method of [JCC06] uses s-FEM [Fis92], a variant of FEM, where two meshes are used simultaneously: a coarse mesh to capture global deformations of the model, and a local mesh activated in a region of interest. The method is limited to small deformations and cannot simulate deformable dynamics. The approach is similar to [JP01], in that it uses Green's functions and capacitance matrices to enable altering the set of vertices with displacement boundary conditions at runtime. Unfortunately, most such haptic rendering papers assume interactions based on pulling on (or constraining) a small set of vertices, not performing real-time deformable object collision detection and distributed contact force and torque computations between two object with complex geometry.

Linear complementary problem (LCP) solvers can be used to solve the Signorini contact prob-

lem between a rigid object and a linear deformable object [DA04, DDKA06]. LCP solvers are computationally demanding which limits the approach to models of modest geometric complexity. In this thesis, we do not use LCP; we use the contact model described in Section 3.5. We demonstrated a deformable pointshell haptic rendering example in [BJ05]; however, one of the two contacting objects (a small rigid ball) had trivial geometry. This thesis supports *both* contacting objects having non-trivial geometry, resulting in a much more difficult collision detection problem. Also contact models based on the use of analytical gradients (as was the case with the ball) give discontinuities if naively extended to non-trivial distance field geometry.

3.5 Penalty-based point-contact model

The point-contact model employed for rigid-rigid, rigid-defo, or defo-defo contact are conceptually identical: the contact penalty forces are determined by querying the points of the pointshell object against the signed distance field of the other object (see Figure 3.6). The user manipulates one of the two objects (the *haptic object*). In our implementation, the pointshell object is anchored (cannot undergo rigid body motion), and the distance field object is the haptic object; the distance field translates and rotates with the haptic object. Alternative configurations where the pointshell object can undergo rigid body motion, or takes the role of a haptic object, are straightforward.

Both pointshell and the distance field are computed in a pre-processing stage (see Sections 3.8 and 3.11). During every haptic cycle, pointshell points are queried against the signed-distance field, as illustrated in Figure 3.6. By convention, negative distance field values denote the interior of the distance field object, and therefore points in contact. Penalty forces are computed for points in contact and no force is assigned to points not in contact. We choose the direction of the force to equal the inward normal of the pointshell point in contact (and as such the force acts to resolve contact; see also Section 3.8). The magnitude is determined from the amount of penetration:

$$F = -k_{\mathrm{C}}\, d\, N, \tag{3.1}$$

where $k_{\mathrm{C}} > 0$ is the contact penalty force stiffness, $d < 0$ is the signed distance field value, and $N \in \mathbb{R}^3$ is the point's inward normal in the world coordinate system. The point world-coordinate position, normal, and distance field value are approximated differently for rigid and deformable objects, and are discussed in detail in later sections. Note that the total force and torque can be computed easily using this model, as can the trilinearly interpolated signed distance field values. The details on virtual coupling, contact torques, and contact force and torque gradient computation are given in Chapter 4. If the number of points in contact ℓ is greater than some constant L (we use $L = 10$), we scale the total contact force by L/ℓ, for reasons of stability. Such scaling is commonly used also in the VPS method (see Section 4.7).

We note that this contact model is not standard, but it has useful approximation properties for haptics. It avoids using the distance field gradients which are discontinuous across the object's interior medial axes. In fact, we first tried using distance field gradients to determine the direction of the penalty force, but the resulting force direction discontinuities gave poor haptic signals. In our

contact model, the force on a point is a continuous function of the object's position, orientation, and deformation. Only the normal of one of the two contacting objects (that of the pointshell object) determines force direction, and this is both an approximation used for speed, and robustness; also note that for smooth surfaces in non-penetrating contact, the normals of the two opposing surfaces at the contact point are colinear.

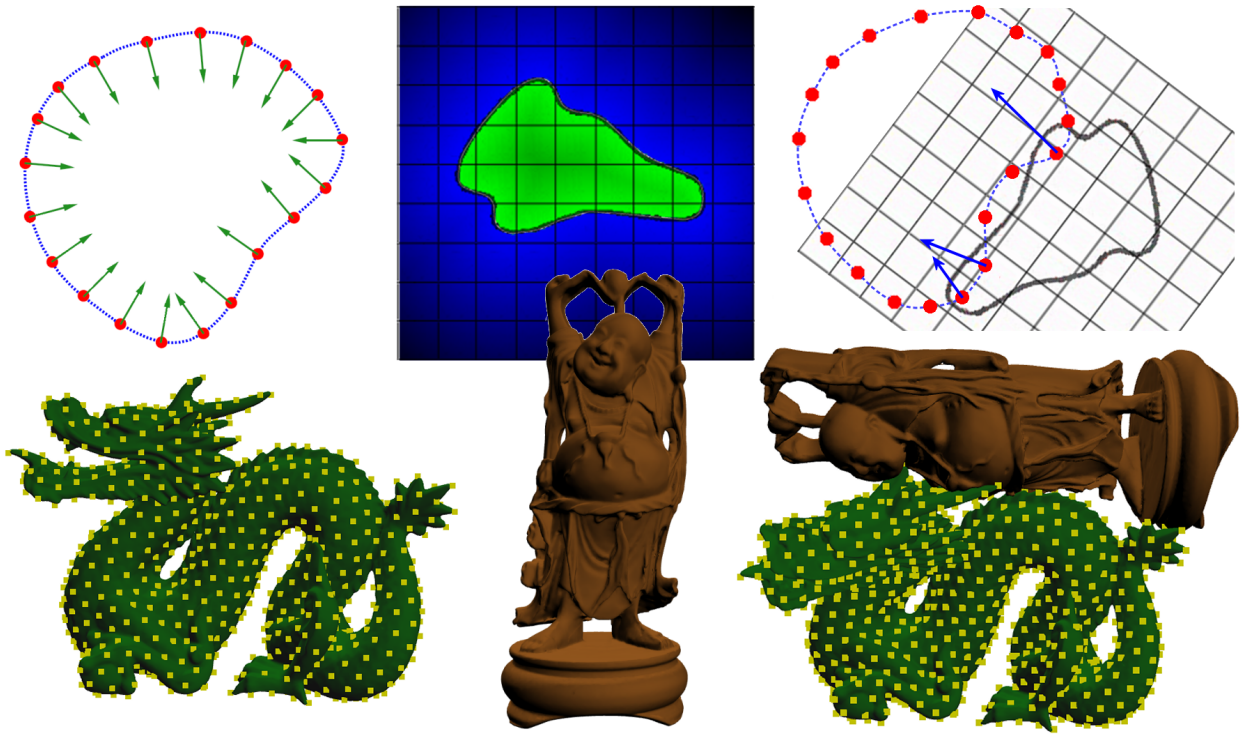


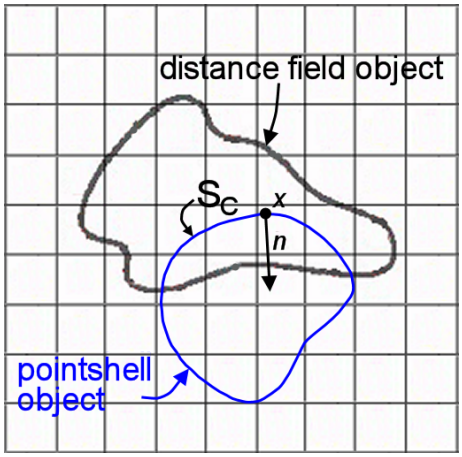
Figure 3.6: **Point-contact model:** Left: Pointshell with inward normals. Middle: the signed distance field. Right: deformable contact and contact forces.

Scaling penalty forces by the area weights of the points (optional step): If a pointshell does not sample a surface near-uniformly, it can be beneficial to scale the penalty force of each point by the local surface area "belonging" to each vertex. Otherwise, if all points are weighted equally, spurious contact torques could occur. For example, this will happen if a U-shaped rigid object contacts (with the two ends of the U) a planar pointshell object, generating two contact sites with about equal area. If one site is sampled to a finer resolution than the other, the U will want to rotate. To alleviate this problem, the weights could be taken to be, for example, surface areas of the Voronoi cell (based on the surface geodesic metric) of each point. These areas could be approximated, for example, by sampling the surface with a large number of randomly positioned point samples, and counting the number of point samples closest to each point of the pointshell. When contact forces are weighted, the quantity computed in Equation 3.1 can be interpreted as

a contact *traction*. Area-weighted tractions give the total contact force on the part of the surface "belonging" to the point in contact.

In our implementations, however, the pointshells come from particle repulsion, which gives near-regular pointshell samplings. Therefore, we do not assign any area weights to points; all points are assigned an equal weight of 1. Also, we keep the weight at 1 regardless of the hierarchical level of the point. This does not cause significant discontinuities if the deepest rendered pointshell level changes from one haptic to cycle to the next because (1) each level is always either rendered completely or not started, and (2) the L/ℓ scaling essentially acts as area weight. For example, if a face of a box is in parallel contact with a planar pointshell surface, and one increases the deepest rendered pointshell level, say, by one level, more points will be in contact, causing ℓ to increase. The scaling L/ℓ will stay proportional to the area "belonging" to each point in contact, so overall contact stiffness will stay approximately the same regardless of the deepest level rendered. This role of L/ℓ is not accidental, as we investigate next.

3.5.1 Contact force convergence under pointshell refinement



In this section, we will prove that our contact force converges to a limit if the pointshell is made progressively finer and finer. In this sense, traversing the pointshell can be seen as collocating the analytical contact model of Equation 3.2 at the locations of pointshell points. Rendering the pointshell at progressively deeper levels provides better and better approximations to the analytical contact model.

To derive the limit, assume that the surface of the pointshell object is in contact with the volume of the distance field object, and denote the part of the pointshell object surface in contact by S_C. Also, assume that the contact surface area is non-zero (otherwise, there is no penetration between the two objects and penalty forces do not apply). As previously mentioned, if the number of points in contact ℓ is greater than some constant L (we use $L = 10$), we scale the total contact force by L/ℓ, for reasons of stability. We are interested in the limit process where the pointshell is increasingly refined to finer and finer levels (say, in an infinite sequence of pointshell levels $\mathcal{L}_1, \mathcal{L}_2, \ldots$). Since the surface contact area is non-zero, the pointshell will eventually be fine enough that more than L points will be in contact. Therefore, it is correct to use the L/ℓ force-scaled version of the contact force in our derivation of the limit.

Theorem: If one samples the surface with an increasingly larger number of near-uniform points (or area-weighted non-uniform points), with the largest gap between the points smaller than ϵ, with

$\epsilon \rightarrow 0$, the total contact force converges to the following integral:

$$F_{\text{contact, analytical limit}} = \frac{-k_{\text{C}}\, L \int_{S_C} d(x) n(x) dS}{\int_{S_C} dS}, \tag{3.2}$$

where $d(x)$ and $n(x)$ are the signed distance value and inward normal at x, respectively.

Proof: Summing the contributions of all points in contact (Equation 3.1) gives the total contact force

$$F_{\text{contact, discrete collocation}} = \frac{-k_{\text{C}}\, L}{\ell} \sum_{i=1}^{\ell} d_i\, N_i, \tag{3.3}$$

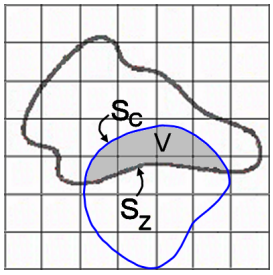
where i runs over all points in contact. Since the pointshell is near-uniform, we can approximate $\Delta S_i = S/m$, where ΔS_i is the surface area "belonging" to point i, S is the total surface area of the pointshell object and m is the total number of points in the pointshell. Then, Equation 3.3 can be rewritten as

$$F_{\text{contact, discrete collocation}} = \frac{-k_{\text{C}}\, L \sum_{i=1}^{\ell} d_i\, N_i\, \Delta S_i}{\sum_{i=1}^{\ell} \Delta S_i}. \tag{3.4}$$

This last equation gives a Riemann sum for the integral of Equation 3.2. That is, Equation 3.4 is an approximation sum for the definite integral of Equation 3.2. As $\epsilon \rightarrow 0$, the Riemann sums converge to the integral by definition (standard definition of integral in calculus). That is, the approximation sums of Equation 3.4 (and hence our contact model of Equations 3.3 and 3.1) converge to the analytical limit given by the integral in Equation 3.2. QED

3.5.2 Our contact model as a volume integral

Our limit contact model of Equation 3.2 has an alternative intuitive representation. First, note that the integral from Equation 3.2 can be extended over the contacting surface area S_Z of the distance field object, as distance field values are zero on S_Z. Now, the integral is over a closed surface and we can use the divergence theorem to rewrite the contact integral as



$$F_{\text{contact, analytical limit}} = \frac{-k_{\text{C}}\, L \int_{S_C \cup S_Z} d(x) n(x) dS}{\int_{S_C} dS} = \frac{k_{\text{C}}\, L \int_V \operatorname{grad} d(x) dV}{\int_{S_C} dS}. \tag{3.5}$$

Our contact limit can therefore be seen as a version of the distance field gradient-based contact model, where each infinitesimal volumetric element on one object feels a contact force proportional to the local distance field gradient on the other object. Note that the accumulated volumetric integral changes continuously when the two objects change position and orientation, even though the distance field gradient is in general not a continuous function. The

contact model is not symmetric if we reverse the roles of the two objects; we discuss this further with our experiments and results (Section 3.13).

3.6 Sampling reduced deformable models

Our approach is designed to work with a variety of reduced deformable models that support a basic two-step simulation process: (1) a fast timestep of reduced deformable dynamics; (2) fast evaluation of individual deformed surface point positions and normals to resolve contact adaptively. Suitable reduced models include classical linear modal vibration models, reduced nonlinear models, simple low-resolution deformable models with embedded geometry, adaptive multi-resolution models, articulated flexible multibodies, etc. The only requirement is that the model's reduced state can be time-stepped at high rates, and its deformed geometry sampled adaptively for time-critical contact force estimation. In our implementation we employ prior dimensional model reduction and Newmark subspace integration techniques for reduced geometrically nonlinear models, as presented in the first part of this thesis. These models are suitable for large deformations with large rotations but small local strain. We make extensive use of the low-dimensional shape parameterization for time-critical collision and force processing. Regardless of the particular reduced deformation model used, our approach addresses stable haptic display of distributed contact between two objects *each* with non-trivial geometry. We also exploit the reduced coordinates, $q \in \mathbb{R}^r$, to perform collision processing (Section 3.8), and to approximate deformed distance field values (Section 3.12).

Without loss of generality (among reduced models) we assume the reduced deformation kinematics $u(t) = Uq(t)$, where $u \in \mathbb{R}^{3n}$ is the displacement vector (e.g., for a FEM simulation mesh with n vertices), $U \in \mathbb{R}^{3n\times r}$ is a certain time-independent low-rank deformation basis matrix, and $q = q(t) \in \mathbb{R}^r$ is the vector of reduced coordinates. The key algorithmic idea is that once the reduced state, q and $\dot{q}$, has been updated by the reduced integrator, co-state quantities such as individual deformable point positions and normals can be sampled adaptively as needed by the tree traversal. This evaluate-when-needed strategy enables us to reconstruct *only* the deformed positions of points encountered during the particular multi-resolution pointshell hierarchy traversal. Points that are culled by the tree need not have their deformed position computed. The traversed set of points is typically a sparse subset of all the points in the pointshell, resulting in significant computation savings. We now describe the evaluation of pointshell points and normals.

3.6.1 Computing deformed point positions

Given the current reduced coordinates q of a haptic cycle, any of the deforming pointshell's S points can be reconstructed in $3r$-flops by summing up the r per-vertex displacement modes:

$$p_s = P_s + U_s\, q, \quad s = 1 \ldots S, \tag{3.6}$$

where P_s is the undeformed pointshell position, and $U_s \in \mathbb{R}^{3\times r}$ is the precomputed modal displacement matrix (interpolated at P_s from U using the FEM shape functions). If the haptic object

is the distance field object, the point then needs to be transformed into the frame of reference of the distance field object (a 4×4 mtx-vec multiply with the same 4×4 matrix for all pointshell points). The signed distance field value d is then obtained by a look up into the distance field (as described in Section 3.11). If the point is in contact ($d < 0$), the normal will be computed, and the penalty contact force evaluated. This force $f_s \in \mathbb{R}^3$ is then applied as an external reduced force $U_s^T f_s \in \mathbb{R}^r$ to the reduced deformable object integrator to excite deformations. This integration timestep is performed at the end of the haptic cycle, producing q and $\dot{q}$ for the next haptic cycle.

3.6.2 Computing deformed pointshell normals

Normals must be computed for sparse pointshell points *in contact* ($d < 0$) for rapid evaluation of penalty forces. The set of points in contact during every haptic cycle is typically small compared to the rest of the pointshell. We need not compute normals for noncontacting points, as the normal is only necessary to determine the direction of the penalty contact force. Rigid pointshells have static body-frame normals that can be precomputed. However, normals change under deformation (see Figure 3.7), and not updating them significantly degrades haptic stability for large deformations.

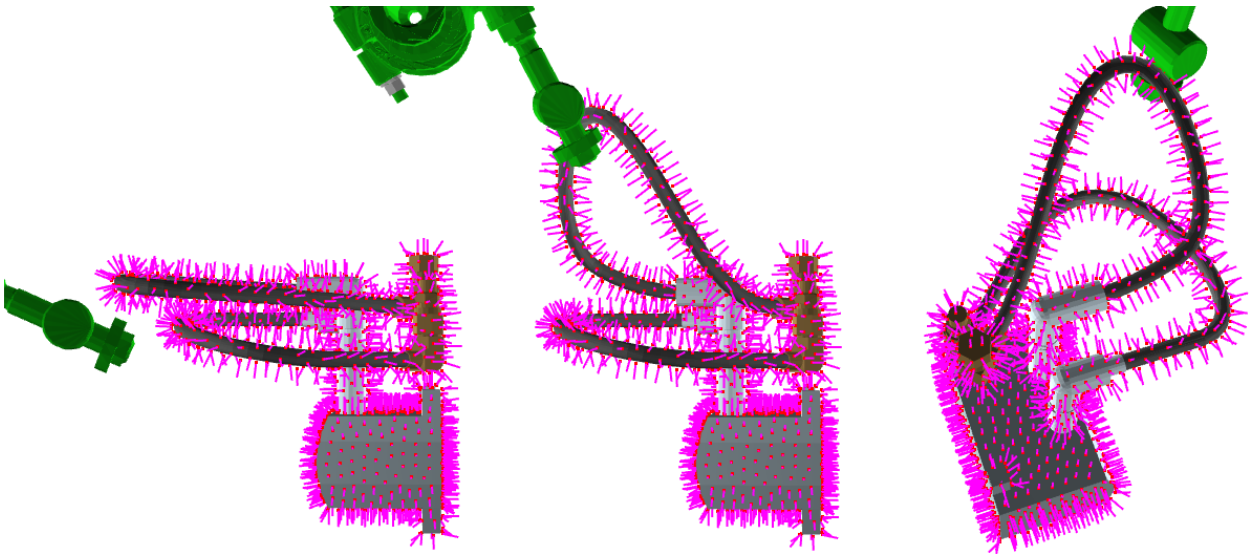


Figure 3.7: **Dynamic normals** (computed using our method): Left: undeformed. Middle, Right: deformed, viewed from two camera angles. In practice, normals need only be computed for pointshell points in contact. Maximum deviation from the correct surface normal is 5.2 degrees in this example; and less than 3.6 degrees for all but five points.

Deforming pointshell normals can be approximated in a variety of ways. Approximating a pointshell point's normal using a nearby surface triangle involves reconstructing at least three points for a cost of $9r$ flops, with weighted surface normals involving more work. Normal is a cross product of two tangent vectors, each of which is a linear function in q, so the normal is quadratic in the components of q (plus normalization). Fast linear approximations involving $3r$

flops (followed by normalization) have been used [JP02] to approximate this quadratic form, but unfortunately their accuracy is restricted to small deformations (and they worked fine for small deformations).

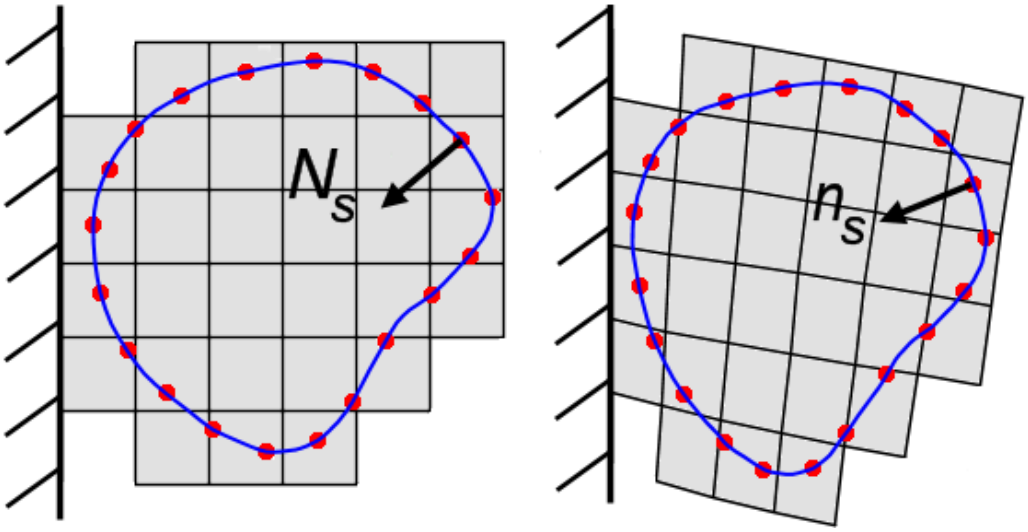


Figure 3.8: **(Inward) normals deformed with deformation gradients:** Left: undeformed configuration. Right: deformed configuration. Our pointshell surfaces are embedded into FEM volumetric meshes (voxel volumetric mesh is shown in gray). Volumetric mesh deformations drive the pointshell point positions and normals.

In our work, we exploit the fact that our geometrically nonlinear deformable models involve large rotations but small strain. This enables us to well approximate the deformed normals simply by transforming the undeformed normal by the local deformation gradient, followed by normalization (see Figures 3.7 and 3.8). The deformation gradient at P_s ($\frac{\partial}{\partial P_s}$ of (3.6)), interpolated from the FEM model, is a linear function of q,

$$F_s(q) = I + \sum_{j=1..r} F_{s,j}\, q_j, \tag{3.7}$$

where s is the index of the pointshell point, and $F_{s,j} \in \mathbb{R}^{3,3}$ can be computed from the deformation basis matrix U and the FEM shape functions. This allows us to construct a fast $3r$-flop pointshell normal approximation,

$$n_s = \frac{n'_s}{\|n'_s\|}, \quad n'_s \approx F_s(q)\, N_s = N_s + \Delta N_s\, q, \quad s = 1 \ldots S, \tag{3.8}$$

where N_s is the undeformed normal, and $\Delta N_s \in \mathbb{R}^{3\times r}$ is a precomputed normal-correction matrix (whose j-th column equals $F_{s,j} \cdot N_s$) that accounts for large rotations–unlike the linearized approximation in [JP02]. Unfortunately, due to its stretch component, the pseudonormal will (a) be (typically slightly) off unit length and needs to be normalized, and (b) will be inaccurate in

the presence of large shear. Nevertheless, we find that the fast pseudonormal is an accurate and haptically stable approximation when deformations consist of large material rotations and small strain. Finally, at double the cost ($6r$ flops + cross product + normalization), true normals could be sampled by instead transforming the tangent space [RJ07]. This would likely not be a significant bottleneck issue; we did not pursue it due to additional code complexity and since the $3r$-flop normals already performed well in our examples.

3.6.3 Alternative approaches to determining the contact normal

In this section, we discuss potential alternative approaches to computing the deformed normals. We considered (and in some cases implemented) these alternatives when designing our algorithm, but eventually decided not to use them. This is because we determined that normals obtained from the deformation gradient (as presented in Section 3.6.2) best match the needs of our simulations, both in terms of speed and continuity: deformation gradient normals are continuous both with respect to contact location and deformations, which is beneficial to stability. Note that all of our strategies obtain the contact normal using only one of the two objects in contact. The rationale for this approximation was discussed in Section 3.5.

Normals from a local point cloud: This approach uses current positions of points in the contacting point's local neighborhood to recompute the normal. Denote the contacting point's position as p_0, and let $p_1, \ldots, p_s$ denote points in some appropriately chosen neighborhood of p_0, appropriately weighted as $w_1, \ldots w_s$, say, according to some kernel centered at p_0. Then, the eigenvector of the smallest eigenvalue of the matrix

$$\sum_{i=1}^{s} w_i (p_i - p_0)(p_i - p_0)^T \tag{3.9}$$

points in the normal direction to the local point cloud [Hop94]. However, this is a costly computation, as measured by the time-constraints of haptic rendering: for every point in contact, it requires computing the current positions of all points p_i ($O(3r)$ cost for each), assembling the 3x3 matrix, and performing Singular Value Decomposition (SVD) on it.

Normals from two (or a few) neighboring vertices: In this approach, one defines a local tangent plane at every contacting point by dynamically computing the vectors from the contact point to two (or a few) pre-selected neighboring points. The normal is then obtained as a cross product of these two vectors (or appropriately averaging cross-products if tracking more than two neighbors). This method can be seen as an approximation to the method from the previous paragraph (normal from a local point cloud). It is very simple, however it requires reconstructing the deformed positions of two additional points, each of which requires a $3 \times r$ matrix by r-vector matrix multiply. This is twice the amount of work of the deformation gradient method that we use in our implementations. Another problem occurs with sharp corners, where it is difficult to define meaningful

local tangents simply by tracking two neighboring vertices. More neighboring vertices might be required in such cases (with higher costs).

Normals from the distance field: The previous two approaches (and the deformation-gradient-based approach used in our simulations) obtain the normal from the pointshell. We also tried an alternative approach of obtaining the normals from the distance field object, but eventually decided not to pursue this approach, for technical reasons described below.

But first, why did we initially try to use the distance field to give us our contact normals? In a simulation with a rigid distance field object and a deformable pointshell object, this would sidestep the problem of the pointshell normals changing under deformation. We could simply obtain the contact normal from the distance field, and not worry about the deformed normal of the pointshell object. Another reason for considering normals from distance field was that if the pointshell object is a thin-shell structure, such as a plastic chair, the normal direction is not well-defined: the surface is infinitely thin and there are as such two candidate (opposite) choices for the direction of the normal. Imagine the thin-shell upper part of the chair in contact with an infinite plane. No matter which of the two directions is chosen for the normal, touching the chair with the ground will pull the chair into the ground for one of the two possible ways of orienting the chair in contact (see Figure 3.6.3). We eventually addressed thin shells and thin wires by fitting pointshell points onto an *offset manifold closed surface* around the thin structure (see Section 3.8.1, last paragraph), but we tried with distance field gradients first, as follows.

The most straightforward way to define normals from the distance field objects is to analytically evaluate the gradient of the distance field at the locations of contact. The distance field is sampled at discrete grid locations inside the distance field box, and can be trilinearly interpolated to arbitrary positions inside a voxel:

$$d(x) = \sum_{i=1}^{8} \phi_i(x)\, d_i, \tag{3.10}$$

where $\phi_i(x)$ is the interpolating function of corner i of a voxel, and d_i is the distance field value in that corner. Such interpolation makes the distance field C^∞-continuous in the interior of a voxel, and C^0-continuous at the voxel boundaries. To obtain the gradient of this distance function, one can analytically compute the gradients of the trilinear interpolation functions, and then evaluate:

$$\nabla d(x) = \sum_{i=1}^{8} (\nabla \phi_i(x)) \cdot d_i. \tag{3.11}$$

The gradients are C^∞ in the interior of the voxels, and not continuous at the voxel boundaries. We implemented this approach, and found it not to work very well. The gradient discontinuities significantly affected the quality of haptic feedback. Namely, distance field gradients change direction discontinuously at the voxel boundaries, which causes the contact force to discontinuously change direction. This discontinuity is worse than with the original VPS, where only the force magnitudes are discontinuous when crossing voxel boundaries, but not directions.

We also considered addressing this problem by precomputing gradients on the points of the distance field grid, and then (trilinearly) interpolating these precomputed *gradients* to arbitrary locations inside a voxel (and then obtain the contact normal via normalization). The force direction is then continuous everywhere, except along the (interior) medial surfaces of the object, where the gradient is discontinuous, and might even flip 180 degrees from one side to the other, causing the contact force direction to flip also. Unfortunately, these interior medial surfaces can reach all the way to the surface of the object. For example, consider a box with 90 degree angles. The medial surface reaches each box edge at an angle of 45 degree to both the adjacent faces. Consequently, severe contact force discontinuities can occur very close to the surface. For these reasons, we decided not to obtain the normal (and therefore contact force) directions from the distance field object.

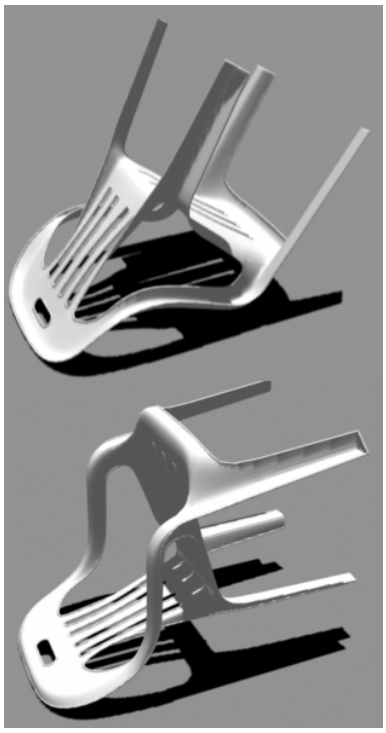


Figure 3.9: **Normals obtained only from the chair geometry** will give contact forces in the wrong direction for one of the two chair orientations.

3.7 Point-based BD-Tree

Our nested tree traversal algorithm (to be discussed in Section 3.10) must be intertwined with deformable collision detection to quickly cull points not in contact. Collision detection is a time-consuming process for general deformable models. With reduced deformable models, it is possible to take advantage of the particular shape model $u = Uq$ to perform collision detection at haptic rates. Such collision detection is possible using the Bounded Deformation Tree (BD-Tree) [JP04], a bounding volume hierarchy equipped with an efficient algorithm to update bounding volumes when the model undergoes reduced deformations (see Figure 3.10).

The BD-Tree method is general in that it can be used, for example, for collision detection against reduced deformable curves, triangle meshes, or tetrahedral meshes. The bounding volume can be spheres, boxes, $k-$DOPs, etc. The strength of the BD-Tree is that it is output-sensitive: unlike most standard bounding volume hierarchy updating algorithms for deformable geometry, it is not necessary to update the entire hierarchy for collision detection. Only bounding volumes for which the immediate parents are in contact will be updated, and this can be done in cost linear in the number of reduced deformable degrees of freedom and independent of geometric complexity. We use a point-based version of BD-Tree with spherical bounds, where the BD-Tree spheres cover the *pointshell points* (as opposed to, say, input triangle geometry), and wherein each sphere center is also a point on the pointshell. That is, our novel nested point tree construction allows us to use actual pointshell point positions as sphere centers, as described in Section 3.8. The

key benefit is amortization: all effort put into updating sphere centers during BD-Tree traversal also simultaneously evaluates deformed pointshell point positions. See Figure 3.11 for an in-simulation illustration of the BD-Tree on one of our haptic examples.

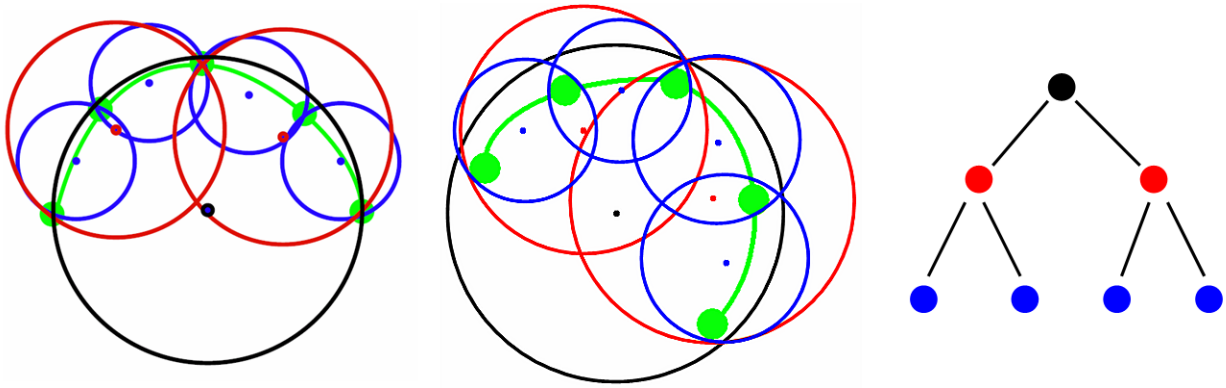


Figure 3.10: **BD-Tree:** Left: The BD-Tree for the green arc geometry in the rest pose. Center: BD-Tree refitted under deformation. Right: the tree topology (which is never updated in the BD-Tree method). In our work, the geometry covered by the spheres are not arcs (as in this illustrative example), but pointshell points.

Only the spheres visited during deformable collision detection need to be repositioned and refitted, and this can be done in r flops, and only using the knowledge of q. It is not necessary to know the deformations of all the points in the pointshell, avoiding the $O(n)$ bounding volume refitting costs of standard collision detection methods. Each sphere center coincides with a pointshell point, so the center is obtained automatically when evaluating the pointshell point position. Undeformed sphere radius, and BD-Tree coefficients that specify how the radius changes under deformations, are all precomputed. In particular, for every BD-Tree node i, the current radius R_i of the bounding sphere can be computed (with r multiplications) as

$$R_i = R_{i,0} + \sum_{j=1}^{r} a_{ij}|q_j|, \tag{3.12}$$

where $R_{i,0}$ is the sphere radius in the rest pose, q_j is the j-th component of the current reduced deformation vector q, and coefficients a_{ij} are time-independent and precomputed (see [JP04] for more details). Radius R_i is not the smallest possible bounding radius; however, it is reasonably conservative (see Figure 3.12). It grows less tight for large deformations, which makes collision detection less effective for large deformations. However, performance was still very good in our examples. We discuss pointshell construction and the point-based BD-Tree in the following sections.

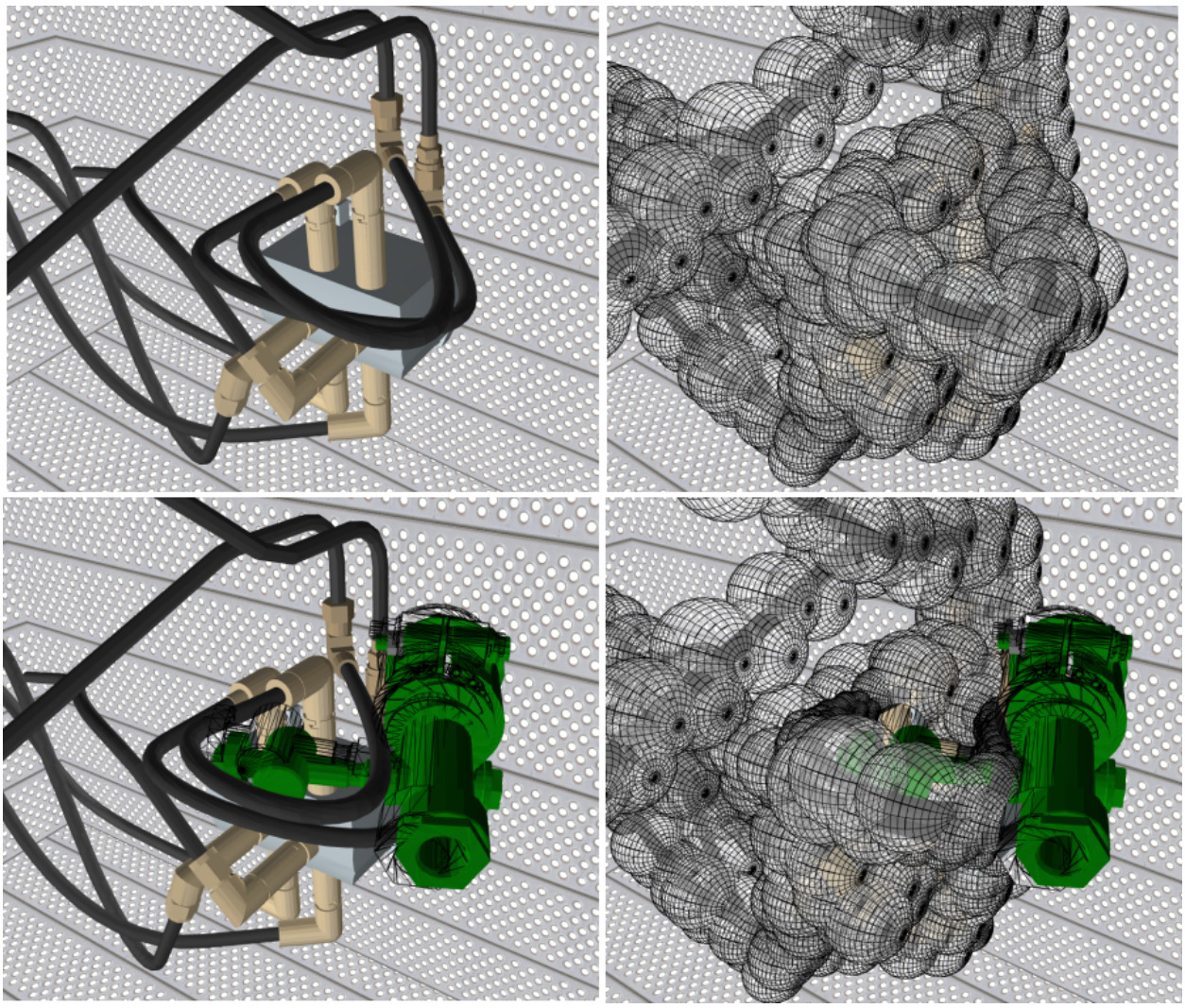


Figure 3.11: **Point-based BD-Tree:** Top row: the rest configuration. Bottom row: tree during contact. Right column shows the deepest bounding spheres that were traversed during this particular haptic cycle (i.e., each rendered sphere is free of contact). Note how the tree is traversed to different depths depending on proximity to contact. We typically use many spheres (320 in this example) on the first level, followed by 4x more spheres on consecutive levels. First-level spheres are always traversed (therefore they appear in top-right image) if temporal coherence is off; however, when temporal coherence is enabled most of them need not actually be checked during most haptic cycles. There are 4 hierarchy levels in this example. The small spheres next to the green metallic part in the bottom-right image are level 3 spheres. Level 4 spheres have zero radius, as they correspond to leaf points.

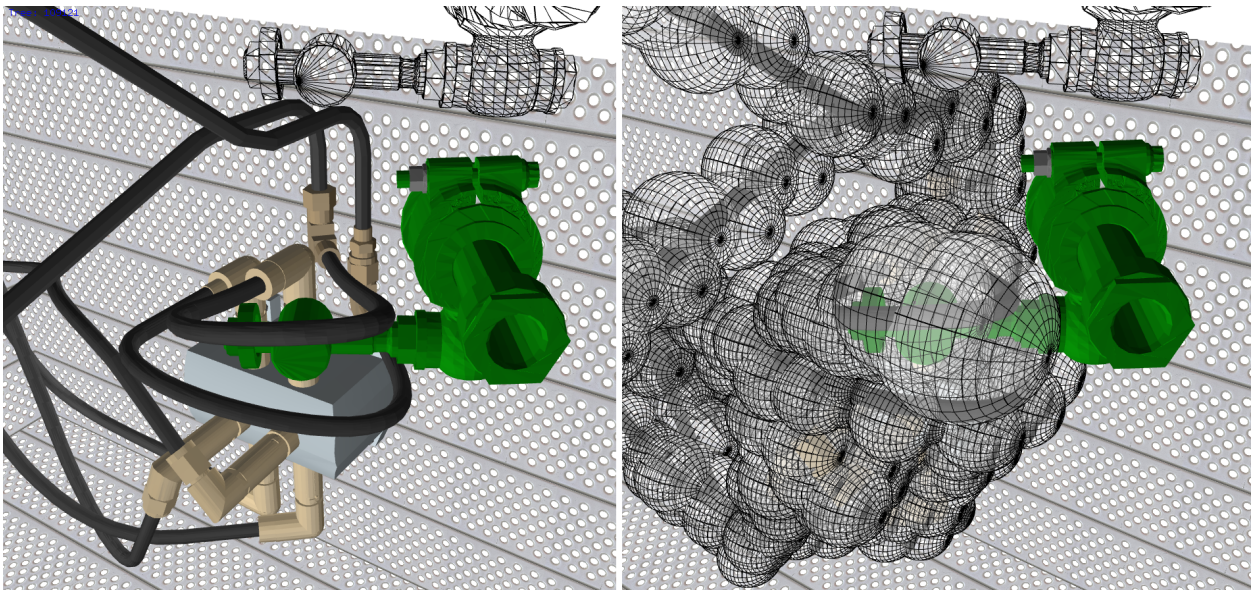


Figure 3.12: **Point-based BD-Tree performance under large deformations:** The sphere radii do grow under large deformations (see, e.g., two level-1 spheres at the tip of the upper u-shaped hose), however tree traversal is still very efficient. The upper hose is bent at an angle of approximately 25 degrees. If the green part is lowered (but kept inserted in between hoses) such that the upper hose returns to (almost) rest configuration, the performance changes from $220\mu s$ to $210\mu s$. To the best of our knowledge, while limited to reduced deformations (as in our case), BD-Tree is the only deformable bounding volume hierarchy update method that can reach haptic update rates (1000 Hz) with example complexity similar to ours. In general, the BD-Tree performs especially well when large deformations are localized to a part of the scene; also, BD-Tree could be augmented by computing bounds at configurations other than the rest configuration.

3.8 Multi-resolution nested pointshell

In this section, we describe pointshell generation. Our pointshells are not just flat collections of points as in the original VPS method. They are organized into a multi-resolution hierarchy, which enables significant computation speedups when large parts of the pointshell are sufficiently far from contact. This section discusses various properties of pointshells and how they are related to/necessary for stable haptic rendering. We then present an algorithm for generating a multi-resolution pointshell guided by these properties.

Uniform pointshell sampling and meaningful pointshell normals are important for reliable contact-force estimates and stable haptic rendering. At first, we considered pointshells sampling polygon soup geometry, but stable haptic rendering cannot be guaranteed due to unpredictable normals and the non-closed and non-oriented nature of the surface sampling. It is not possible to guarantee stable haptic rendering of penalty-based contact forces for arbitrary pointshells: imagine the rendering chaos if pointshell was a closely packed set of points, randomly positioned in a unit

box with randomly assigned normals (see Figure 3.13). The pointshell must also sample a closed and oriented surface; otherwise, the inside of the object can enter contact, and/or the normals and consequently contact forces could point in the wrong direction. Consequently, we designed our pointshells to sample a *closed manifold oriented surface*. When combined with virtual coupling saturation (Section 4.9), this restricts the distance field object to lie outside the pointshell object (or at most a shallow penetration). A similar problem happens if points are too far apart from each other, allowing a small feature in the distance field object to slip undetected in between two points (and then abruptly being pulled out if it touches the pointshell from the interior side). This problem can be prevented by keeping the pointshell resolution equal or finer to that of the distance field. The original VPS pointshell was obtained by voxelizing the pointshell geometry, and using centers of surface voxels as points of the pointshell. This approach is easy to implement and requires few parameters, however we found that it can result in a relatively large number of points, many of which are redundant. The VPS pointshell essentially samples the contact surface with points on an axis-aligned grid, resulting in "staircase" sampling of the surface. We found that pointshells that are positioned on an actual *smooth closed manifold surface*, and (quasi-)uniformly sample that surface, tend to be stable; high frequency components, either in point locations or point normals tend to cause instabilities. Sharp surface corners can still be rendered by properly orienting the normals, albeit not aggressively.

3.8.1 Multi-resolution pointshell via particle repulsion

We generate our pointshells by fitting a set of *particles* (points) onto a smooth manifold surface, using the ideas of particle repulsion [Tur92, WH94]. Similar to [Tur01], we construct a nested point hierarchy by fitting progressively finer samplings of points ($4\times$ branching factor), while freezing coarser level points. We, however, fit points on a level set of a distance field as opposed to a triangle mesh (see Figure 3.14). We do so since (1) it was convenient to re-use our distance field code, (2) the distance field abstracts all underlying geometry and as such pointshell generation runs in time independent of the complexity of the underlying triangle mesh, (3) "polygon soup" geometry (common in virtual assembly simulations) can be supported.

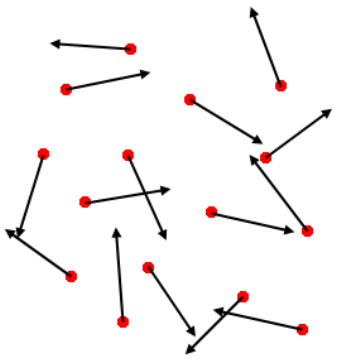


Figure 3.13: **Unstable pointshell:** This "nightmare" pointshell will result in a very unstable (chaotic) haptic signal.

In particular, the pointshell generation proceeds as follows. We fit the particles onto a level set of a distance field. This distance field is not to be confused with the signed distance field of the other object (the distance field object) in the simulation. If pointshell object is given as a closed manifold triangle mesh, we compute a signed distance field to the pointshell object geometry, and use the zero-distance level set. If it is given as a triangle "soup", we compute an unsigned distance field, and use the level set of some small positive distance value. In either case, call the resulting

level set surface $\mathcal{T}$. We also construct a triangulated level set surface $\mathcal{S}$ using marching cubes. For unsigned distance fields, we also remove any connected surface components lying completely inside another component. In our examples, we identified the connected components using a simple greedy algorithm. Then, we manually identified the exterior components and removed the interior components; this last step could also have been automated. We used single precision floating-point distance fields with resolutions ranging from $128 \times 128 \times 128$ (8 Mb of storage) to $512 \times 512 \times 512$ (515 Mb of storage (a $513 \times 513 \times 513$ grid)). Our distance field computations took on the order of 1 hour for $256 \times 256 \times 256$ resolutions and meshes with about $100,000$-$200,000$ triangles. These computation times could be shortened using advanced distance field computation techniques [SOM04, SGG+06b]. We computed the distance fields everywhere inside the bounding box (on grid vertices). Note, however, that these distance fields are only required to obtain $\mathcal{S}$ and to constrain the particles to the level set surface $\mathcal{T}$ during repulsion. Therefore it is sufficient to only compute the (unsigned) distance field in some appropriate shallow layer surrounding the geometry, as opposed to everywhere inside the distance field box. Such computation could proceed faster and require less memory.

Our nested hierarchy construction is similar to [Tur01]. We first "inject" the n_1 points of the coarsest level 1, by using stratified area-weighted random sampling of $\mathcal{S}$ [Tur92]. We then let the particles repel each other while constraining them $\mathcal{T}$. We do so by alternating between particle repulsion and constraint stabilization (to $\mathcal{T}$). Our repulsion potential among two particles is radial, and its gradient equals $2(d/R)^3 - 3(d/R)^2 + 1$, where R is the finite influence radius, and d is the Euclidean distance among two particles. Gradient is set to zero for $d \geq R$. At every iteration step, we assemble the repulsion gradients, then move each particle a distance of $-\eta$ in the direction of the net gradient, where $\eta > 0$ is the descent parameter. This step will move the particle (slightly) off the level set surface $\mathcal{T}$. We restore it back to $\mathcal{T}$ by performing a Newton solve using the distance field and its gradients: we walk in the direction of the gradient of the distance field, until the particle is within ϵ tolerance to the level set value. This two stage process is repeated until particles have converged sufficiently. We then permanently fix the position of the particles, and inject n_2 new particles that will form level 2. These new particles are influenced both by level 1 particles and among each other. Again, the level 2 particles are iterated until convergence, when we fix them, and insert level 3 particles. This procedure is repeated until a desired number of levels, or a desired sampling accuracy, has been reached. We use $n_{i+1} = 4n_i$ so that surface sampling resolution about doubles with every level. We also decrease the particle repulsion radius R by a factor of two for each new level. The resulting multi-resolution pointshell has the property that for every level k, the set of points up to level k samples the level set surface $\mathcal{T}$ about uniformly (see Figure 3.14). Note that the triangle surface $\mathcal{S}$ is only necessary to be able to efficiently draw initial random positions of particles. Computing the gradient of each particle by traversing all other particles is expensive, so we accelerate it using a hash table. Every particle can then quickly traverse the particles inside neighboring cells, up to the necessary distance given by the finite repulsion radius.

The repulsion algorithm uses parameters R, η, and ϵ. We tuned these parameters manually, by trial and error. We set ϵ to a constant small value; we typically used $\epsilon = 10^{-4}$ with our scenes (total size of scene geometry was typically between 1 and 10 units). Repulsion radius R and descent

parameter η were tuned by running the repulsion interactively on coarse repulsion levels, with the user tuning the two parameters until pointshell visually looked good quality (near-uniform point distributions). Too high η causes oscillations in point locations, too low η causes slow convergence. Too high R causes points to spuriously accumulate at sharp edges, too low R causes points to stall, as they are no longer affected by their neighbors. The choice of R, η is important as repulsion does not perform well with suboptimal values of parameters, but works really well with good parameter choices. We were always able to find good values for R, η within a few tuning iterations. Then, we used these parameters to run an offline process that generated as many pointshell levels as desired, splitting R by 2 for every new level generated. It typically took several hours to fit our pointshells with $256,000$ points onto a $256 \times 256 \times 256$ unsigned distance field. Other than the hash table, the process was not optimized. If desired, particle fission could be used to iron out local point irregularities [WH94], however, such irregularities were rare in our examples.

Pointshell points need to be assigned (inward-pointing by convention) normals. We do so by locating the closest triangle of $\mathcal{S}$, and assigning the normal to be the pseudonormal [BA05] of the closest feature (face, edge, or vertex). Other variations include using barycentrically weighted pseudonormals, the normalized gradient of the distance field, or, if original geometry is a closed manifold mesh, the pseudonormal of the closest feature on that mesh, which best preserves sharp corners.

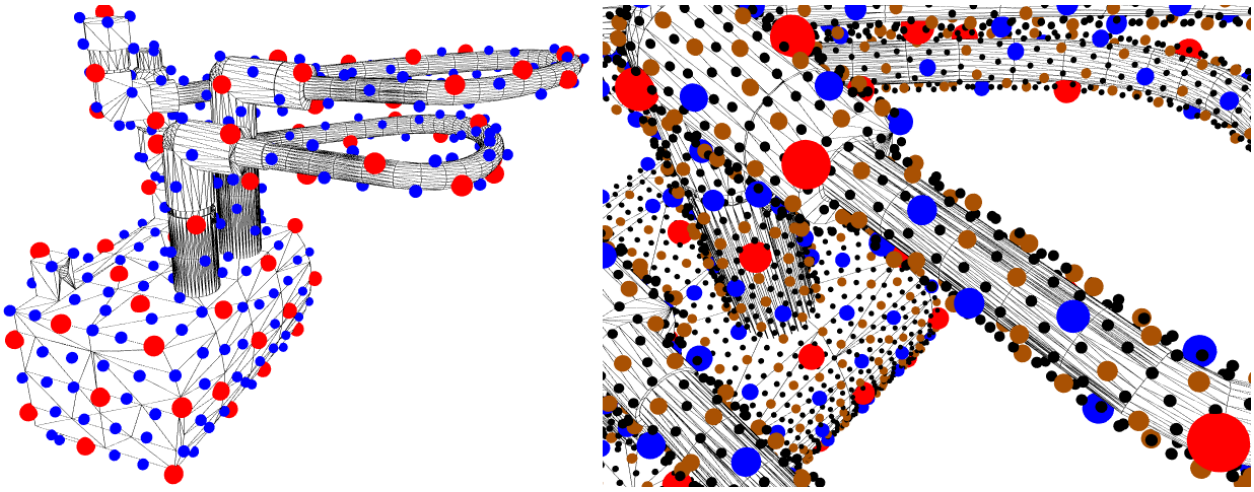


Figure 3.14: **Multi-resolution oriented pointshell from particle repulsion:** Left: The first two levels of a part of the hose scene. Right: detail with four levels shown. Points are fitted on a small-distance offset oriented manifold surface to support "polygon soup" input geometry, and provide oriented surface normals for contact.

Simulating thin shells and thin wires: This same approach of fitting points on a small-offset level set of a distance field also allows us to simulate thin shells and thin wires, by treating them as pointshell objects. No modification of the pointshell generation algorithm is necessary. Extracting

the isosurface of the unsigned distance field effectively serves to construct an offset surface for the thin structure. Pointshell points are fit onto this offset surface. Normals on opposite sides of this offset surface then oppose one another, giving correct contact force directions. The choice of the offset value is important, since the pointshell must be thick enough to prevent pop-through, and at the same time thin enough to still approximate thin structures well. We were able to find such offset values in practice. The thin structure must be resolved well, and therefore high resolution distance fields are typically required (e.g., 256x256x256 or higher). For example, this is how we simulated the cables of the deformable pointshell bridge (Figures 3.15 and 3.16). See also the discussion in Section 3.6.3 on thin-shell structures.

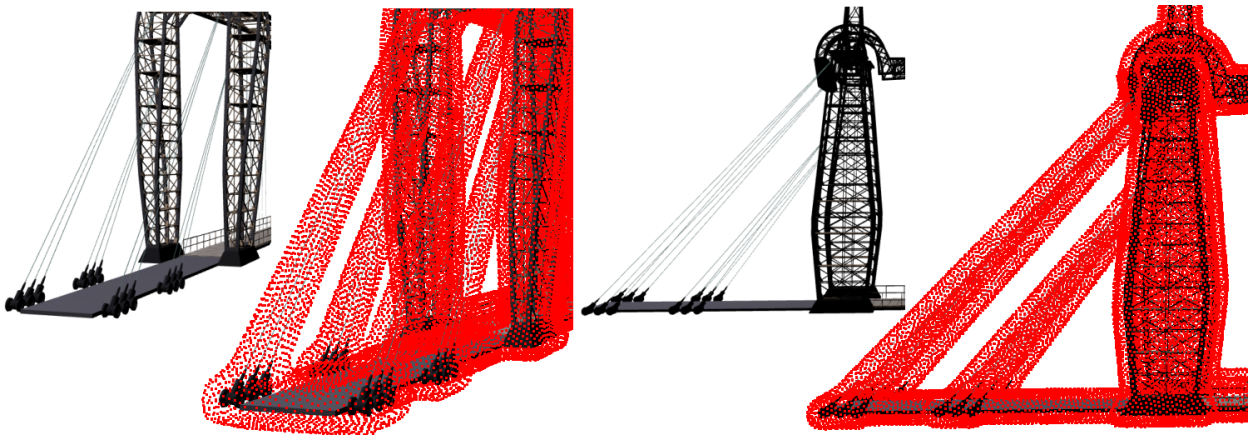


Figure 3.15: **Pointshell for thin structures (e.g., cables):** First and third image from left: the bridge geometry, viewed from two camera angles. Second and fourth image: the pointshell. Each red dot is one point. Points are fitted onto a level set (offset surface) of an unsigned distance field to the bridge geometry. Distance field resolution was 256x256x256 for this pointshell, and offset distance was 1.4x the size of a distance field voxel.

3.8.2 Pointshell generation alternatives

In this section, we briefly describe some alternative approaches to pointshell generation. These approaches could be used instead of the particle-repulsion generated pointshells. Points could then be organized into a multi-resolution hierarchy using a general-purpose hierarchy generation method (described in Section 3.9.1). However, in this thesis, we advocate the particle-fitting approach as the general method of choice because it can provide (near-)uniform sampling, at all resolution levels. All the other methods described below will typically give less uniform pointshells, and/or they will not aim at organizing the pointshell into a hierarchy, requiring a general-purpose hierarchy generation method, which sometimes results in suboptimal hierarchies.

The first approach that we tried in our research was implementing the pointshell generation technique of the original VPS method. In this method, centers of voxels intersecting the input polygonal geometry are used as the pointshell. In the VPS method, point normals are assigned from

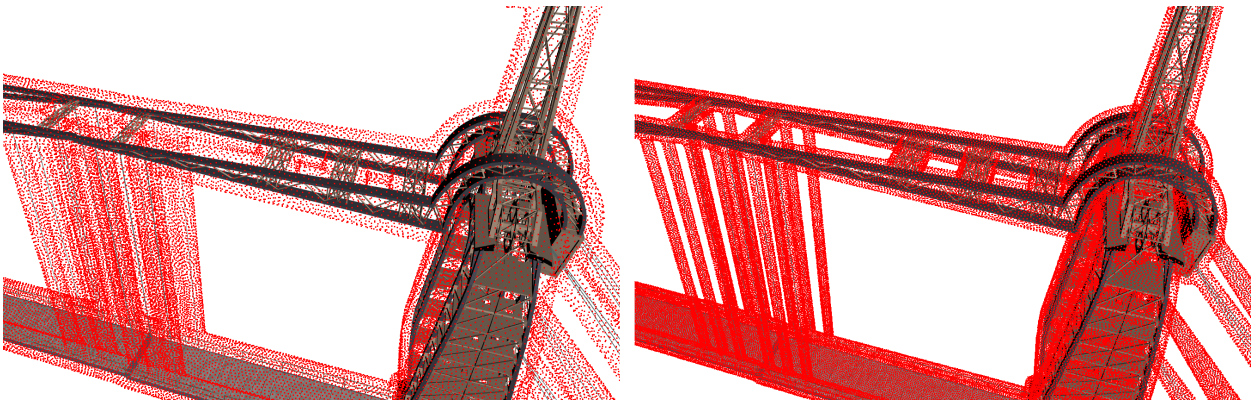


Figure 3.16: **Pointshell for thin structures with a smaller offset:** Left: pointshell (four levels, 85,000 points) fitted using a 256x256x256 unsigned distance field (same pointshell as in Figure 3.15). Right: pointshell (five levels, 256,000 points) fitted using a 512x512x512 unsigned distance field, and with a 2x smaller offset distance as pointshell on the left. Both pointshells use an offset of 1.4x the size of the distance field voxel. Note that the finer pointshell resolves all twelve central vertical cables on the bridge, whereas the coarser pointshell merges groups of three cables together. Simulation with the finer pointshell is more permissive (accurate) in that thicker objects can be inserted through holes in pointshell geometry, such as the holes in the bridge's upper shelf. In general, higher resolution unsigned distance fields are required for smaller offsets, otherwise the offset surface becomes progressively non-smooth (noisy).

the nearby voxel geometry, or from suitable nearby triangle geometry (personal correspondence with Boeing). We simply located the nearest triangle to each pointshell point and assigned the pseudonormal of the nearest feature (vertex, edge or face) on this triangle. In our experiments, these "classical" VPS pointshells often resulted in redundant samplings of the contact surface. For example, in areas of high surface curvature there were regions where the surface was represented by two "layers" of points (with normals pointing in about the same direction). Since the point locations originate from a voxelization, the surface is sampled along a rectangular grid (which is effectively high spatial frequency content). In our experiments, the VPS pointshell was reasonably stable. However, stability improved when we started using pointshells from particle repulsion, especially at (generally more difficult) higher stiffness settings.

Another approach that we tried is to simply use the vertices of the input geometric model as the pointshell points. This will give poor results if the model mesh vertices do not sample the surface near uniformly. For example, large triangles will introduce under-sampled regions. If triangle sizes vary across the model, bias will be introduced into contact forces, since more densely populated regions will contribute larger contact forces. One fix to this problem is to sample randomly positioned points on the triangles (with randomization weighted by triangles' surface areas), as opposed to vertex positions. However, pointshell uniformity cannot be guaranteed and will be moderate at best in practice. Another possible solution is to use contact weights for points, as discussed in Section 3.5, but this is still not a good solution; the randomness of point positions

will adversely affect the haptic signal, and point resolution will be wasted in regions that ended up being densely sampled by the random number generator.

Another approach to generate the pointshell is the *Layered Depth Cube* algorithm [PZvBG00] from point-based rendering. This algorithm was designed for converting polygonal models into a point representation suitable for point-based graphical rendering. In this approach, the input polygonal surface is intersected with rays of parallel lines. Three bundles of rays are used: one for each of the x, y, z coordinate axes. Within each bundle the rays are parallel to the coordinate axis, and follow a regular grid pattern. Points are defined as intersections of rays with the geometry (see Figure 3.17). The resulting sampling will have the property that, roughly speaking, every location on the input surface is at most a distance of $h\sqrt{3}/2$ away from some point sample, where h is the distance between parallel rays. However, the sampling is non-uniform due to the bias introduced through the chosen x, y, z directions. For example, three points will be positioned close to each other at any location where the surface is close to the intersection of three orthogonal rays. This might not be a major issue for point-based graphical rendering, but in haptics, it can lead to instabilities in the force/torque signals.

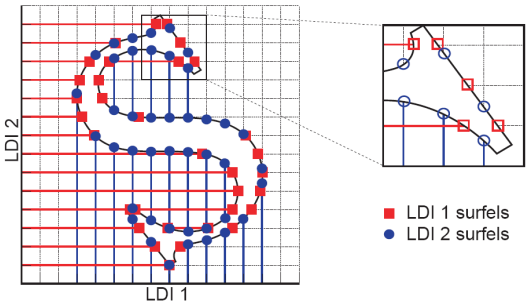


Figure 3.17: A point cloud generated using the Layered Depth Cube algorithm.

Both the random point sampling strategy and the Layered Depth Cube strategy result in non-uniform point distributions. Particle repulsion arises very naturally in this context, as it allows to smooth out these irregularities. This is what eventually led us to use particle repulsion as the pointshell generation method of choice in this thesis. As we described in Section 3.8.1, we use random point sampling to initialize point locations, followed by repulsion. Another option would be to use the LDC point cloud as the initialization step.

3.9 Nested point-tree construction

In this section, we demonstrate how the multiresolution pointshell can be organized into a *nested point-tree*, a novel datastructure that combines a nested multiresolution pointshell with a deformable sphere-tree hierarchy (BD-Tree). Only small pointshells (~ 4000 points on our hardware: Intel Core 2 Duo 2.66 Ghz processor with 2 Gb RAM, manufactured in 2007) can be traversed linearly (point by point) in a haptic cycle. To some extent, larger pointshells are possible by using temporal coherence and wider voxels [MPT06]. To simulate much larger pointshells (rigid or deformable), we organize the pointshell into a nested tree hierarchy (see Figure 3.18). The (deformable) pointshell hierarchy also serves as a (BD-Tree) bounding sphere hierarchy. Such a nested point-tree essentially converts the tree traversal into a linear operation, which in turn enables graceful degradation of contact.

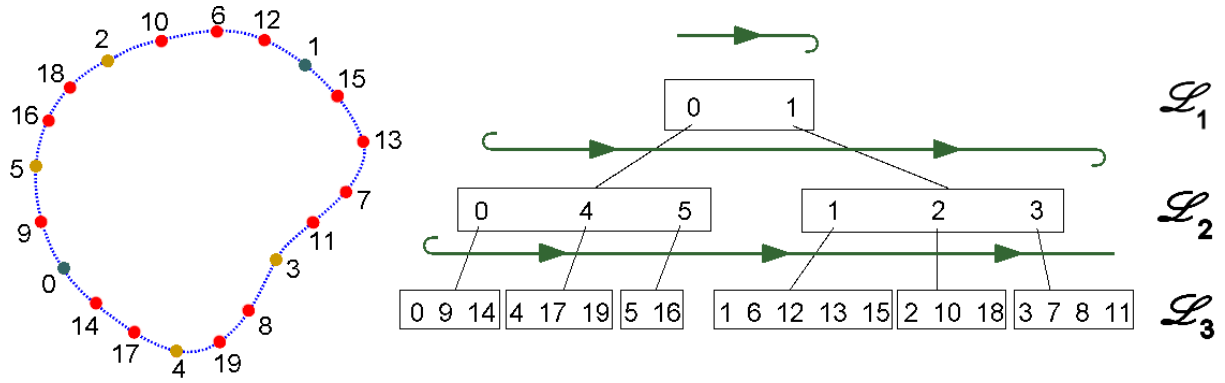


Figure 3.18: **Nested point-tree:** Left: The pointshell. Right: the hierarchy, the traversal order and tree levels $\mathcal{L}_i$. Particle-repulsion levels are 0-1, 2-5, and 6-19 in this case.

After particle repulsion, each point is positioned, and assigned to one of the disjoint *particle-repulsion* levels. We proceed by organizing the points into a sphere-tree hierarchy. In the deformable case, this results in a point-based Bounded Deformation Tree (BD-Tree). We first generate *tree levels* $\mathcal{L}_i$: the set $\mathcal{L}_i$ consists of an instance of every point of particle-repulsion levels 1 through i (see Figure 3.18). Therefore, if a point first appears on particle-repulsion level i, an instance of this point will appear in all tree levels $\mathcal{L}_j$, for $j \geq i$. Instances can share common memory data. For example, when traversing the tree at runtime, the distance field value for a parent can be cached and reused when instances of the same point are traversed further down the hierarchy. The deepest tree level consists of a single instance of all the points in the pointshell. The elements of $\mathcal{L}_i$ are our tree *nodes* on tree level i. This construction was chosen so that all nodes at all levels lie on the actual contact surface $\mathcal{T}$, and that each $\mathcal{L}_i$ samples $\mathcal{T}$ about uniformly (without gaps at locations of coarser level points), with progressively finer resolution with level index i. This is in contrast with previous time-critical bounding volume hierarchy approaches in literature [Hub95, MO06]. In these previous approaches, the contact surface is resolution-dependent, consisting simply of a union of bounding spheres (with progressively smaller radii depending on the level-of-detail).

We establish the tree parent-child relationships by traversing nodes at every level $\mathcal{L}_{i+1}$, and assigning each node to the nearest node from level $\mathcal{L}_i$. We call the resulting tree hierarchy a *nested point tree hierarchy*. A bounding sphere radius is computed for each non-leaf tree node, centered at the location of the node, and covering all nodes in the subtree, in the undeformed configuration. The radius of the bounding sphere gets progressively smaller for instances of the same point at different levels. For deformable pointshells, the sphere radius is updated at runtime using appropriate precomputed BD-Tree datastructures (update cost is r flops per traversed node). The sphere center always coincides with the current node location (and its child instances).

3.9.1 Nested point-tree construction for alternative pointshell generation approaches

The "alternative" pointshell generation techniques of Section 3.8.2 only give the positions of the points. In order to benefit from a multi-resolution pointshell, it is necessary to organize the points into a multi-resolution nested tree. Here is one general algorithm to do so. First, pick a "seed" point and denote it p_0. The seed can be picked, for example, at random. Find the point furthest away from p_0 and denote it p_1. Next, p_2 is defined to be the point with the maximal minimum distance to $\{p_0, p_1\}$. This process is repeated, p_k being the point with the maximal minimum distance to $\{p_0, \ldots, p_{k-1}\}$. When all the points have been selected, the algorithm has ordered the pointshell points into a sequence. Next, we organize the points into a tree, as follows. We first select a branching factor B for the tree (we typically used $B = 8$). The first B points in the sequence are defined to be level 1 points, the next B^2 points are level 2 points, and so on until all points have been assigned to a level. These levels then take the role of particle-fitting levels, and from here on, nested point-tree generation proceeds exactly as described in Section 3.9. We note that the idea of sorting point locations by selecting the next farthest point is quite common; for example, in [ELPZ97], they use it for progressive transmission of 2D images.

3.10 Time-critical progressive contact forces

We now describe the core approach for time-critical evaluation of contact forces when one object is represented as a rigid or deformable pointshell, and the other object provides a signed-distance oracle, $d(p)$.

3.10.1 Runtime tree traversal

At every haptic cycle, the algorithm traverses the pointshell hierarchy in breadth-first order (see Figure 3.18). For every tree node, the algorithm looks up the value of the signed distance field at the node's location. Next, to support deformations, the BD-Tree method is used to update the radius of the bounding sphere covering points in the subtree rooted at the node. If the distance field value is greater than the radius, no point in the subtree can possibly be in collision, and the subtree is not traversed further (see Figure 3.19 (Bottom, node 1)). Otherwise, all children of the node are added to a list for future traversal (see Figure 3.19 (Top, node 0)). If the distance field value is negative (contact), a penalty force is assigned to the node, just like in the single-resolution case. If descendant copies of the node (i.e., same point on deeper levels) are visited during later traversal, no additional force is accumulated. The copies, however, are not skipped from traversal (unless graceful degradation forces the traversal to terminate before the corresponding level is reached) since they might have proper (i.e., non-copy) children in contact. Also, the distance value is cached the first time a point is looked up into the distance field in a given haptic cycle, and subsequently simply read from memory when visiting descendant copies.

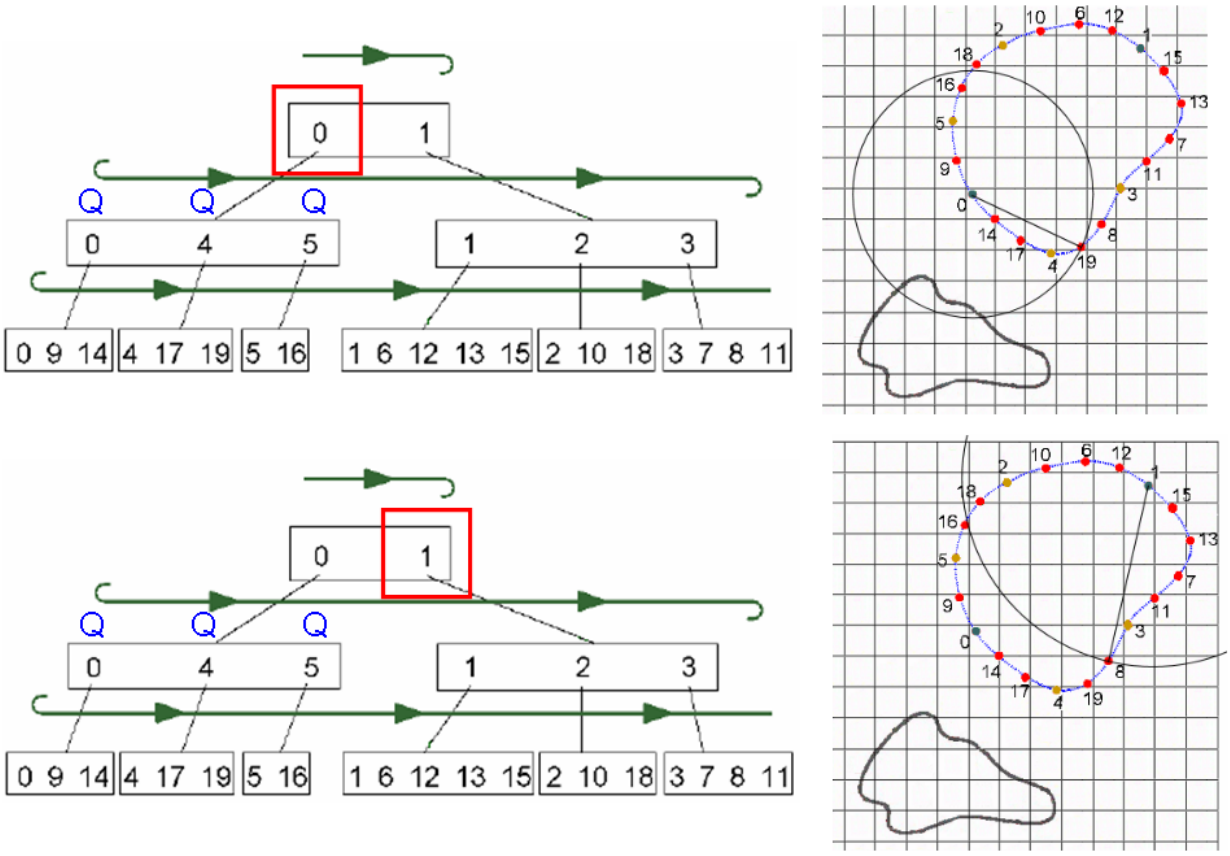


Figure 3.19: **Tree traversal:** Left: The bounding sphere of node 0 intersects the environment and all the children are queued for traversal (indicated by the symbol 'Q'). Right: Node 1 is sufficiently far from contact that its bounding sphere does not intersect the environment, and as such, the subtree need not be traversed further. This completes the traversal of Level 1. The traversal list for Level 2 is $\{0,\ 4,\ 5\}$, where 0 is in this case the copy on level 2.

The subtle but algorithmically key point is that our point-based BD-Tree uses nested pointshell points as deformed sphere centers. This seemingly trivial choice enables time-critical testing of the point-based BD-Tree against a distance field: updating a BD-Tree sphere also updates a deformed pointshell point, p, which can in turn be immediately evaluated against the distance field, and if in contact ($d(p) < 0$) will contribute to the progressive accumulation of contact forces in a multi-resolution manner. Therefore, even in the expected case where there is insufficient time to completely test the BD-Tree against the distance field, approximate contact forces can still be accumulated in a time-critical fashion.

3.10.2 Temporal coherence

Temporal coherence [Hub95, MPT06] is exploited using space-time bounds as follows. When a tree node is processed, an estimate of the earliest possible haptic cycle when the bounding ball at this node can enter contact is determined (and cached with the node), based on node's current distance to contact, sphere radius, and maximum point velocity that can occur in the simulation. This cycle can be the immediate next one if the node is close to contact (or in contact), or it can be hundreds of cycles into the future for points far from contact. Next time a node is about to be processed, we can safely skip it if the current haptic cycle is younger that what stored in the node. Temporal coherence reduced our simulation times by 20%-45%.

We currently do not take into account pointshell point deformation velocities in our temporal coherence estimates (our deformations are low-frequency). We only use the haptic object maximum linear and rotational velocities. Including pointshell deformation velocities would be straightforward, with the additional cost of having to construct point velocities ($\dot{p_s} = U_s\dot{q}$, see Equation 3.6 on page 109). It would also be necessary to impose a maximum acceleration limit on $\ddot{q}$ (and take this into account in the temporal coherence estimate). This could be done inside the implicit Newmark integrator, in a way similar to how we limit haptic object velocities and rotational velocities (Section 4.6). In particular, after the implicit Newmark integrator has evaluated the current internal forces and damping forces (and before solving the $r \times r$ linear system), one can compute the current reduced acceleration as $\ddot{q} = -\tilde{D}(q, \dot{q}) - \tilde{R}(q) + \tilde{f}$. If $\|\ddot{q}\|_2 > \ddot{q}_{\max}$, then one can scale (for this timestep only) $\tilde{D}(q, \dot{q}), \tilde{R}(q), \tilde{f}$ by $\ddot{q}_{\max}/\|\ddot{q}\|_2$. Such scaling will only chop high accelerations. These are not very typical of manipulation scenarios; they can occur for example, if the user deliberately quickly pushes on the deformable object, or if the object is very stiff with strong internal elastic forces. They can also occur if the user strains the deformable object and then suddenly releases it. This technique is general and can be used whenever one needs to keep the velocities or accelerations of a reduced deformable object within prescribed limits.

3.10.3 Graceful degradation

A desirable property of a haptic rendering algorithm is *graceful degradation*: if there is not enough computation time to complete the contact computation fully, the algorithm should not fail, but rather return a reasonable approximation, given the limited computation time available.

If the contact scenario is not very involved, much of the tree will be culled quickly and only a small fraction of the pointshell will be traversed. In contrast, if there are many different contact sites, such as when the pointshell object is positioned close to the distance field object, a large fraction of the pointshell will need to be traversed, preventing a complete tree traversal within one haptic cycle. To support graceful degradation in such cases, we traverse the tree progressively level by level (see Figure 3.20). While traversing the list of nodes that need to be visited on level i (as determined by level $i-1$), we build the traversal list for level $i+1$. At the end of level i, we compare the size of this list to a measure of the remaining computation time for the haptic cycle. Only if enough time remains, we render the next level. The algorithm always either

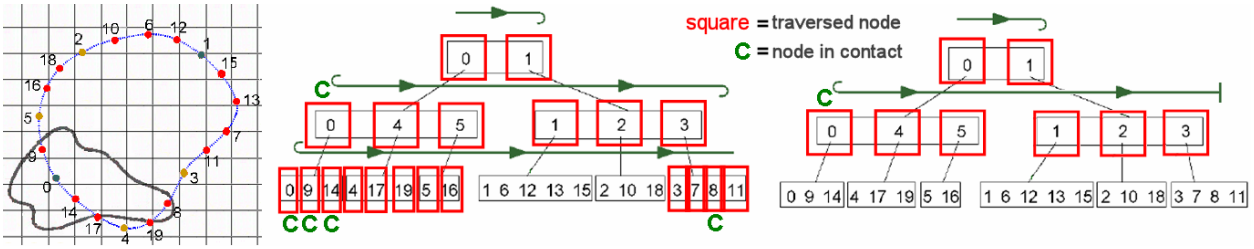


Figure 3.20: **Graceful degradation:** Left, Middle: With involved contact scenarios, many tree nodes need to be traversed during a complete hierarchy traversal. Right: Under graceful degradation, tree is traversed only up to a shallow depth, as permitted by available CPU power.

completes a level or does not start it. The resulting contact forces are identical to those obtained under an exhaustive tree-less traversal of points up to a given level. Note that such a scheme is of course only approximate: only geometric features resolved by points on the deepest traversed level will be rendered. The benefit of our algorithm is that rendered surface resolution and contact force accuracy both increase with available computing power, as deeper levels can be rendered. The level at which the traversal terminates depends on the difficulty of the contact configuration and the cold and warm thresholds. Faster computers can use larger thresholds and will as such render deeper levels (potentially the entire pointshell), i.e., more accurate contact forces will be computed. With progressively finer pointshells, contact forces converge to a limit, as demonstrated in Section 3.5.1.

LOD control: The amount of work required to process a node is predictable [1], and it is not necessary to time the execution explicitly. Instead, we count the total number of tree nodes visited, V, during the current haptic cycle. We use two thresholds to determine whether to render the next tree level. All the levels up to and including the deepest level rendered in the previous haptic cycle use the *warm threshold* V_W, whereas all deeper levels use the *cold threshold* $V_C < V_W$. A level is rendered if the total number of tree nodes visited before entering this level is below the level's threshold. Cold threshold is stricter to prevent popping back and forth among two levels during consecutive haptic cycles; we set $V_C = 0.8V_W$. In practice, we have seen LOD changes at most about 2-3$\times$ per second, and did not feel any popping haptic rendering artifacts. LOD changes can be made fewer by reducing V_C.

Warm threshold selection: In our simulations, we selected V_W manually by running the simulation and observing computation times per haptic cycle. Often, we would use $V_W = 2500$ for a new (2007) Intel Core 2 Duo 2.66 Ghz processor with 2 Gb RAM, and $V_W = 600$ for a three-year-old (2004) 3.0 GHz dual Intel Xeon processor with 2.75 Gb of memory. The selection of V_W was not difficult and could be automated, as the cost to process each node is $O(r)$, establishing a high correlation between the number of traversed nodes and the total haptic cycle computation

[1] Applies also to deformable distance fields, if the proxy shell is small and spatially nearly uniform, such that the kNN search runs in approximately constant time.

time. For example, the user could set the desired maximum computation time per haptic cycle. The computer could then (continually) measure the execution time using the standard system timers, reducing or increasing the warm threshold to meet the desired performance.

Counting points in contact: Another possible improvement to our thresholding scheme would be to count both the number of points traversed and the number of points in contact. Namely, points in contact require more work than traversed points not in contact. Traversed points require $3r$ flops to construct point position, r flops to construct BD-tree radius, and $O(1)$ flops to transform the point to the frame of reference of the distance field object (4×4 mtx-vec multiply) and look up the signed distance value. Points in contact additionally require $3r$ flops to construct the contact normal, one sqrt computation to normalize it, $O(1)$ work to update the total contact force and contact force derivatives, and $3r$ flops to project the contact force into the reduced deformable basis. Therefore, we could predict the computation time approximately by a properly weighted linear combination of the number of traversed points $V_{\mathrm{traversed}}$ and number of points in contact V_{contact}. That is, we would not render the next level if the predicted next level completion time

$$\alpha\Big(V_{\mathrm{traversed}} + V_{\mathrm{queued}}\Big) + \beta\Big(V_{\mathrm{contact}} + \frac{V_{\mathrm{contact}}}{V_{\mathrm{traversed}}} V_{\mathrm{queued}}\Big) \tag{3.13}$$

is greater than the appropriate (warm or cold) threshold. Here, $V_{\mathrm{traversed}}$ and V_{contact} refer to the node count at the end of the current level, and V_{queued} refers to the number of nodes queued for traversal on the next level. Note that the number of nodes in contact on the next level is not known. The above prediction estimate is based on the ratio between the nodes in contact and traversed nodes up until the current point in time. Given the cost of operations for points in contact, β could be set to 1.5α. This is, however, probably too pessimistic, as the $O(1)$ distance field look up costs (memory access, trilinear interpolation) come with a relatively high constant. However, in our implementation we opted for the simple count-traversed-points-only scheme (as described in the LOD control paragraph). The number of points in contact is typically small compared to the total number of traversed points. Our experiments indicate that the amount of time spent in the point-contact part of the code is small compared to the point-traversal code; but of course, this becomes less so in scenarios where the ratio between points in contact and traversed points is high. In such cases, a scheme such as the one from Equation 3.13 might be employed to predict next level computation times more accurately.

Low-pass filtering geometry at coarser levels: It might be advantageous to have the different LODs sample low-pass-filtered versions of the geometry, as opposed to one and the same surface at all levels. The motivation for this is that if the surface has high-frequency spatial components, sampling it near-uniformly at coarse resolutions might result in aliasing in contact forces. We did not address this extension in this thesis, and were still able to generate very stable haptic simulations with reasonably detailed models. The problem might be more pronounced with extremely detailed objects. Note that the issue is more pressing with normals than with point positions. Normals are essentially spatial derivatives of geometry and are as such more prone to aliasing than the

geometry itself. One potential extension of our work would be to generate the pointshell hierarchy in the same way as in this thesis, but assign different normals to copies of the same point at different levels. Each normal would be assigned with respect to the appropriately low-pass-filtered version of the pointshell object geometry.

3.11 Rigid distance field

A rigid distance field object in our simulations carries a *signed* distance field, $d(p)$, against which any transformed pointshell point p can be queried to detect contact ($d(p) < 0$). Computing a *signed* distance field is straightforward for closed manifold meshes. In the common case of "polygon soups" we first compute an *unsigned* distance field to the geometry, then extract a level set isosurface for some small positive distance value using marching cubes to obtain a closed manifold mesh [LLVT03] (see also [SOS04]). Next we remove any mesh components completely contained inside another component, and then compute a *signed* distance field with respect to the remaining closed manifold mesh(es). At runtime, we interpolate the distance field grid values to the query location if the query location is inside the distance field box. If it is outside the box, we conservatively estimate

$$d(p) \geq d_{\text{box}} + d_{\text{boundary}}, \tag{3.14}$$

where d_{box} is the shortest distance from the query to the distance field box, and d_{boundary} is the query-independent minimum distance field value on the surface of the distance field box (see Figure 3.11).

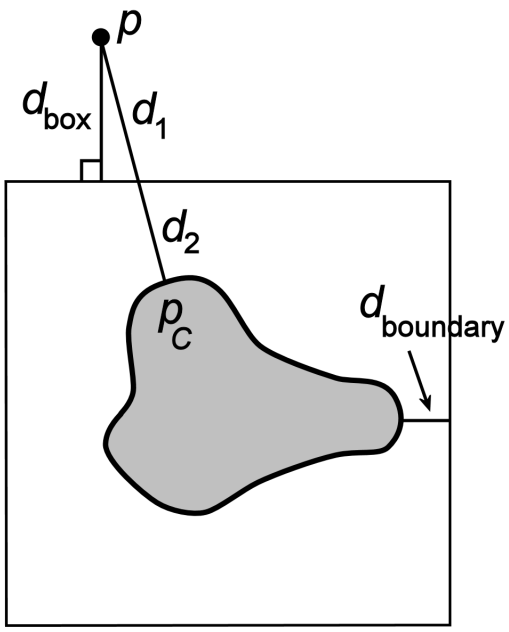


Figure 3.21: **The proof of Inequality 3.14.**

Inequality 3.14 has an important practical value: many pointshell points (when transformed to the frame of reference of the distance field object) will not lie inside the distance field bounding box, because they are far away from contact. Typically, the distance field bounding box will cover some reasonable amount of space encompassing the distance field object; but making the distance field box too large is impractical since it either leads to a larger distance field memory footprint, or to lower resolution in resolving the features of the distance field object (since a lot of the resolution is "wasted" in free space). In our implementation, we first fit a tight centered box (with all sides equal, i.e., a cube) to the distance field object, then scale the box from its center by a certain expansion factor (typically 1.5). Without Inequality 3.14, when p is outside the distance field box, all that can be said about $d(p)$ is $d(p) \geq d_{\text{boundary}}$. When traversing the nested point-tree, we would therefore have to assume the worst case $d(p) = d_{\text{boundary}}$. If p is very far away from the box, this

estimate is wasteful. For example, if p is an inner node in the nested tree, its bounding sphere might be deemed to be potentially colliding, and as such its children will be traversed, even though in reality p and the bounding sphere for its subtree might be far away from contact. In contrast, Inequality 3.14 provides an estimate that *grows* with the distance of p to the bounding box: for points very far from the bounding box, the estimate will be dominated by the d_{box} term. Tighter (and hence larger) distance estimates also enable more aggressive temporal coherence estimates of point's shortest time to contact. Note that the distance estimate discontinuity when crossing in/out of the distance field box does not affect the validity of temporal coherence predictions, as our distance estimate is always conservative: it reports a value smaller than the true distance to contact.

The proof of Inequality 3.14 follows by connecting the query point p to its closest point p_C on the surface (see Figure 3.11). The line segment from p to p_C penetrates the distance field box boundary at exactly one location: if it penetrated the box boundary at two locations, the second location would make the line segment exit the box, so a third penetration would be necessary to re-enter the box; no straight line, however, can intersect the box boundary in more than two locations (except if the line lies in the plane of a face of the box; but then, it cannot connect to a closest point inside the box). Call the distance from p to the bounding box intersection d_1, and let d_2 be the distance from the bounding box intersection to p_C. Then, $d(p) = d_1 + d_2$. We must have $d_1 \geq d_{\text{box}}$, since d_1 is the length of *some* segment joining p and the distance field box. Also, we must have $d_2 \geq d_{\text{boundary}}$, since d_2 is the length of *some* segment connecting a point on the box boundary and a point on the object's surface. Therefore, it follows that $d(p) = d_1 + d_2 \geq d_{\text{box}} + d_{\text{boundary}}$. QED.

Comparison to VPS voxmap: We also implemented the standard VPS voxmap [MPT99] and found distance fields to provide more stable haptic feedback, especially at higher stiffness settings. We attribute this difference to discontinuities in voxmap contact force when crossing voxel boundaries. Distance fields also better support deeper penetrations: voxmap force vanishes as soon as the single-voxel contact layer is penetrated, whereas the distance field continues to give a repulsive force even for deep penetrations. The distance field contact force is proportional to penetration, and only peaks when reaching the object's interior medial surface. To reduce distance-field memory, a hybrid data structure would be possible, using a wide-bit voxmap in free space, and a distance field accessed via a hash table [SL06] in the shallow contact force layer surrounding the object. With increasing distance field resolution, the number of surface voxels (where 32-bit floating point values must be stored) grows quadratically, whereas the number of all voxels (where n-bit voxmap must be stored, $n < 32$) grows cubically, so such hashing becomes more and more advantageous with increasing resolutions.

Fast runtime evaluation of distance fields: Given a query location p inside the distance field box, we evaluate $d(p)$ using trilinear interpolation of precomputed distance field values at the vertices of the rectangular grid. However, in order to perform the trilinear interpolation, it is first necessary to find the distance field voxel containing the query location p, i.e., it is necessary to

compute

$$\Big(\text{floor}(\frac{x-x_0}{h}),\ \text{floor}(\frac{y-y_0}{h}),\ \text{floor}(\frac{z-z_0}{h})\Big), \tag{3.15}$$

where the distance field box is assumed to be aligned with the coordinate system axes, (x_0, y_0, z_0) is the lower-left-bottom corner of the distance field box, h is the distance field grid spacing, and $p = (x, y, z)$. It is possible evaluate Equation 3.15 directly, say, using the $(\text{int})(\_)$ float-to-integer type-cast in C. However, on Intel architectures, (int) compiles into an instruction that causes the CPU pipeline to stall temporarily. We were able to obtain a 30% speedup in overall distance field lookup using an alternative approach that avoids explicit integer rounding by exploiting the structure of the IEEE floating point format [Nin02, Eri01]. This optimization is possible when the distance field resolution is a power of 2. Essentially, if $w \in [0, 1]$, and $[0, 1]$ is divided into 2^k uniform buckets, then the index of the bucket containing w is given (in binary notation) by the first k mantissa bits of $1 + w \in [1, 2]$. Furthermore, the remaining mantissa bits give barycentric coordinate within the bucket, which can then be used directly for trilinear interpolation. We wrote C inline assembly code that uses these ideas to quickly find the voxel containing the query point and barycentric weights within the voxel.

3.11.1 Analytical implicit functions instead of a distance field

The haptic simulation can be simplified if the voxmap or distance field is replaced by a simpler implicit function with an analytical formula, such as a sphere, cube, or combination of infinite planes (see Figure 3.22). Analytical gradients are then continuous (potentially with some singularities) and can be used to determine the contact normals. The contact force direction will be discontinuous when crossing the medial axis (Section 3.6.3). The singularities are, however, isolated or even non-existent for very simple objects such a sphere or a single plane. Such analytical implicit function simulations are, of course, limited by the fact that analytical implicit functions cannot model detailed geometry.

3.12 Deformed distance field

Our real-time contact algorithm performs over a thousand signed-distance-field evaluations per millisecond-long haptic frame. Distance fields for complex rigid objects can be preprocessed and runtime-sampled quickly, however the same is not true of the deformable distance fields needed to support time-critical "defo-defo" contact. In this section, we explain how we can exploit reduced deformations to provide approximate signed distance values very quickly.

If distance field computation (evaluation) costs were zero or negligible (which is not the case today, but perhaps possible in the future), one could simply treat the deformed distance field as an oracle to provide the exact distance $d(p)$ to be used in the contact force model of Equation 3.1 on page 105. On today's hardware, however, distance field evaluation for deformable models is expensive and cannot be performed at haptic rates for non-trivial geometry [SGG+06b, SGG+06a].

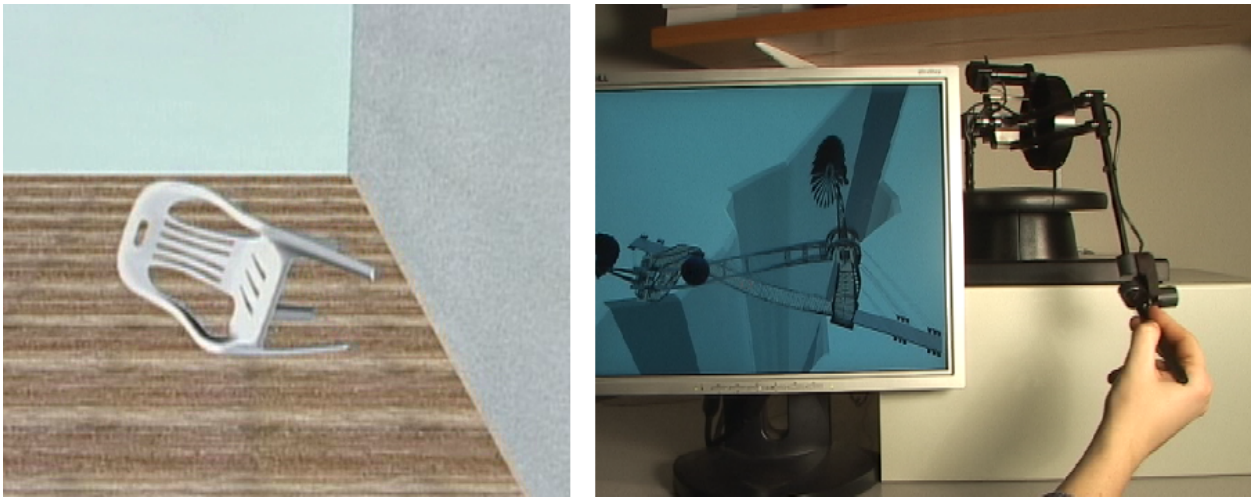


Figure 3.22: **Haptic contact between reduced deformable objects and rigid objects modeled by simple analytical functions:** Left: The user-manipulated deformable chair is a deformable pointshell with about 26,000 vertices, rendered against an analytical implicit function of two perpendicular walls. Right: The deformable bridge is a deformable pointshell of about 41,000 vertices, rendered against a user-manipulated sphere. The chair is a linear reduced deformable model (linear modal analysis deformations). The bridge is a nonlinear StVK reduced deformable model, as described in the first part of the thesis.

Even if an acceleration datastructure such as a bounding volume hierarchy was computed (refitted) to the deformed triangle mesh before each haptic cycle, each individual point distance query requires $O(\log(n))$ search time, where n is the number of triangles in the model.

Deformed distance fields can be computed on *grids* in appropriate regions of interest using methods such as Fast Marching [SM01], or 3D scan conversion [SPG03] (also see [JBS06] for a survey). Fast marching has also been applied to unstructured locations inside the object [MAC04], however the approach relies on good spatial distribution of the distance field query locations, with the locations known in advance at the beginning of each haptic cycle, neither of which is the case in our simulations. These methods recompute the distance field from scratch, i.e., they are general-purpose distance field computation algorithms, invoked separately with each deformed configuration. They cannot update the distance field at hard real-time rates for complex geometry as their computation times depend on the number of model triangles; the workload required to process each haptic cycle would also grow with the distance field resolution.

A deformed distance field can be approximated by re-using a precomputed distance field in the object rest configuration. One such "material-space distance" approach was proposed by [HFS$^+$01], and it works as follows. First, note that deformable objects typically carry a deformable simulation mesh, such as a tetrahedral mesh or hexahedral mesh (this is also the case in our work). One can then maintain a bounding volume hierarchy on top of the tetrahedral mesh. Given a query location p, one can use the hierarchy to quickly locate the tetrahedron containing p, or detect that

p is not contained in any tetrahedron. If inside a tetrahedron, one can use p's barycentric weights to pull p back to the rest configuration, obtaining the corresponding rest configuration location P. A distance value estimate is then obtained by looking up P into the precomputed signed distance field. There are two main problems with this method. First, the computed distance value will be inaccurate under stretching and compression. Second, the methods cannot report distances outside of the object, i.e., it is restricted to the domain of the tetrahedral mesh. The first problem can be alleviated by using a *push-forward* to the current configuration [TSIF05]. After computing P, one does not use the precomputed undeformed distance field directly. Instead, a Newton solve across the undeformed distance field is performed, starting from P and guided by the distance field gradients, to obtain a location S on the zero signed distance field level set $\mathcal{S}$. Point S serves as a reasonable approximation to the closest point to P on $\mathcal{S}$. Then, one can use another precomputed static bounding volume hierarchy to quickly locate the closest point to S on the object's rest configuration triangle mesh (call this closest point S_M). Note that this triangle mesh can be much higher resolution than the tetrahedral mesh. Finally, the deformed distance is approximated as the distance from p to the *current* (i.e., deformed) location of S_M.

In our work, we only need the distance at the sparse locations of traversed tree nodes, which only become known progressively during the traversal. Also, we need to be able to compute distances quickly; both [HFS$^+$01] and [TSIF05] were designed and used in offline simulations. Material-space methods do not assume any particular structure to the query locations, and as such permit convenient output-sensitive random access to the signed distance field. The disadvantage of material-space methods is that one needs to maintain a bounding volume hierarchy on the object's deforming simulation mesh, and traverse this hierarchy for each deformed signed distance field query. Because we typically have to perform many distance queries in each haptic cycle, this method cannot sustain haptic rates for objects with complex geometry. We propose a variant of the material-space distance method which does not require a deformable bounding volume hierarchy and which can also compute distance approximations for query locations outside the object. During pre-process, we compute a signed distance field with respect to the undeformed configuration. At runtime, we compute fast approximations to the deformed distance field values $d(p)$ of a reduced deformable object, at the locations of traversed pointshell points. We do so by pulling pointshell point positions back to the undeformed configuration of the distance field object, followed by a lookup into the precomputed distance field, similar to [HFS$^+$01], but with some key modifications. During pre-process, we fit a small pointshell (typically ~ 40 points) to the surface of the *distance field object*. We call this pointshell the *proxyshell* and its points the *proxies*. We fit the proxies using the same algorithm that was used to fit the points to the surface of the pointshell object (Section 3.8.1). Proxies deform together with the deformable distance field object. To evaluate the deformed distance field at some query pointshell point location x, we first perform a k-nearest neighbor search to locate the k closest proxies $p_1, \ldots, p_k$ to x (see Figure 3.23, Left, Middle). We then assign non-negative weights to p_i; closer proxies are given larger weights, and the weights

sum to one; we use the scheme described in [BBM+01] to assign weight w_i to proxy i,

$$w_i = \frac{w_i'}{W}, \qquad w_i' = \frac{1/d_i - 1/d_{\max}}{1/d_{\min} - 1/d_{\max}}, \qquad W = \sum_{i=1}^{k} w_i', \tag{3.16}$$

where $d_i = \|x - p_i\|_2$ and $d_{\max}$ ($d_{\min}$) is the max (min) of d_i over $i = 1, \ldots, k$. We typically set $k = 5$ in our implementations to avoid singular configurations where $d_{\min} = d_{\max}$ (five points in general do not lie on the same sphere in 3D).

We use local deformation models at p_i to generate an approximation to the deformed distance field at x, as follows. Let P_i be the material position of p_i; then the deformation transformation in the vicinity of P_i can be approximated as

$$X \mapsto x = p_i + F_i(X - P_i), \tag{3.17}$$

where X is an arbitrary material point sufficiently close to P_i, and $F_i \in \mathbb{R}^{3\times3}$ is the deformation gradient at P_i. We evaluate F_i in $9r$-flops using (3.7), and the precomputed deformation gradient mode matrices, $(F_i)_j$. We then pull the query position x back to X_i, by inverting equation 3.17 (see Figure 3.23, Right). Next, we obtain d_i, the distance with respect to proxy i, by looking up the undeformed field at X_i. Our distance approximation is $d = \sum_{i=1}^{k} w_i d_i$. Note that the weights are continuous, and always vanish at the furthest nearest neighbor, ensuring C^0 continuity of d both with respect to x and the reduced coordinates of the deformable distance field object.

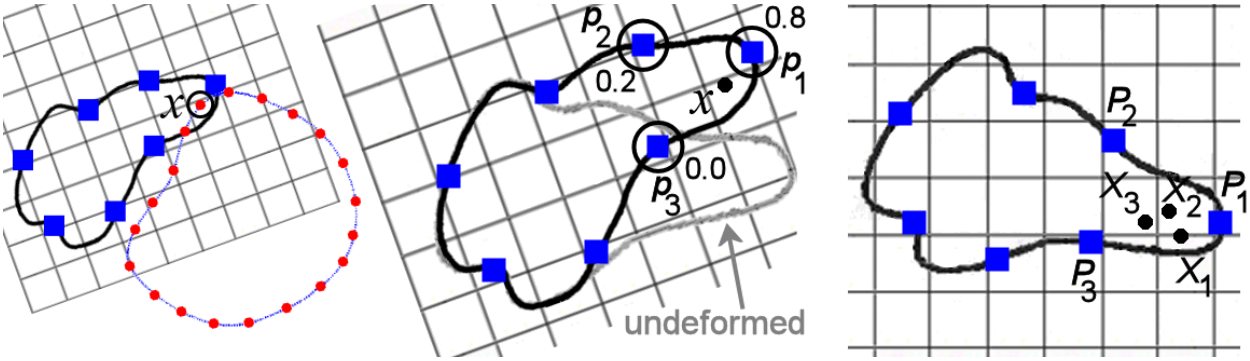


Figure 3.23: **Deformed distance field approximation:** Left: Proxies (squares), the query pointshell point at x. Middle: 3-nearest neighbors and their weights. Right: material configuration pull-backs X_i of x.

The resulting contact forces are applied (with opposite sign) to both the pointshell reduced deformable object and the distance field reduced deformable object. The latter requires projecting the contact force into the subspace of the reduced-deformable distance field object. In order to project with respect to the correct $3 \times r$ modal submatrix, it is necessary to identify where on the distance field object is the location of application of the contact force. One could perform an $O(n)$ nearest-neighbor search, for example, against the deformed positions of distance field

object simulation mesh vertices. For performance reasons we approximate the modes at the contact location with the modes at the nearest proxy (which was already identified during the k-nearest neighbor query).

Our scheme is capable of computing a fast approximation to the deformed distance field with respect to *complex* geometry. This is possible because the complex geometry is encoded into the distance field in the undeformed configuration. The k-nearest neighbor search is fast since it is performed only on a small number of proxy points. Our approximation benefits from spatially smooth modal basis deformation fields, and becomes progressively less accurate under stretching/compression.

3.12.1 Multiple domains

The basic deformable distance field scheme can suffer from spurious contact regions when two nearby parts deform away from each other: "mirror" images of one part appear in estimates d_i under proxies on the other part (see Figure 3.24, Middle). These artifacts can be reduced by dividing the distance field object into several domains, precomputing a separate distance field for each domain, performing the single-domain scheme with respect to each domain, and taking the minimum (see Figure 3.24, Right). Mirroring essentially introduces "extra" geometry, which typically pushes the isocontours of the approximate field further out than the true isocontours, leading to conservative distance estimates. Strictly speaking, however, we cannot guarantee conservative distance estimates. Although more costly, deformable distance field accuracy might be improved under compression deformations by pushing forward material-space closest-point information [TSIF05].

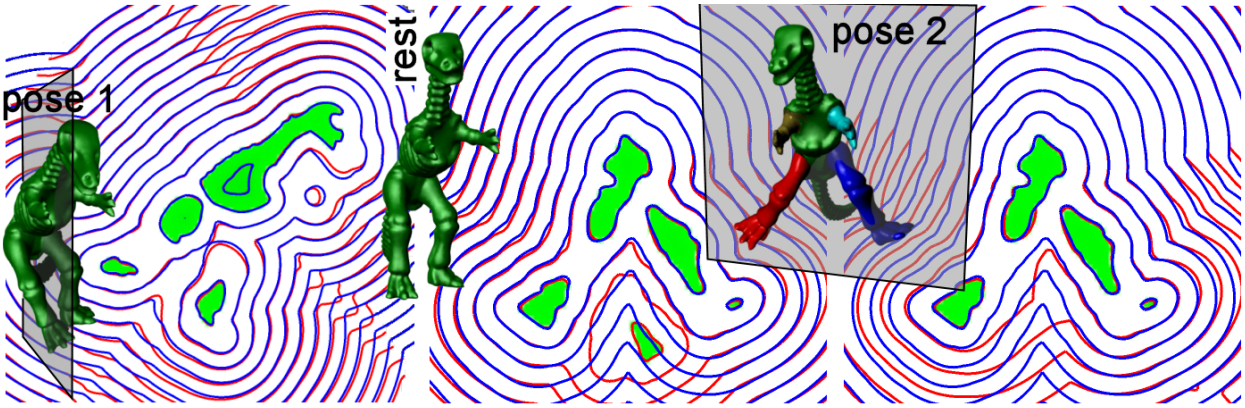


Figure 3.24: **Isocontours of the exact and approximate deformed distance field:** Left: Pose 1, single-domain approximation. Middle: Pose 2, single-domain approximation with the two legs mirroring. Right: Pose 2, multiple domain approximation without mirroring. Green denotes the interior of the object, with respect to the approximated field.

3.13 Experiments and results

Contact case; pointshell+distance field	Contact depicted at:	Pointshell		Dist. field resolution	Traversed nodes		Points in contact	LOD	Timings [μs]				
		levels	points		with TC	TC off			VC	StVK	TT	total	TL
knight(r) + axe(r)	Figure 1.3	6	1.02M	256	2050	3400	820	6	90	0	780	870	80,000
CAD scene(d) + metallic part(r)	Figure 1.4	4	80K	128	530	1520	50	4	90	45	125	260	6,600
dragon(d) + Buddha(r)	Figure 3.6	5	256K	256	1750	2900	650	5	90	45	615	750	29,800
dragon(d) + dinosaur(d)	Figure 1.3	5	256K	256	300	1500	40	3	90	90	660	840	590,000

Table 3.1: **Model statistics** for selected contact configurations, timed on an Intel Core 2 Duo 2.66 GHz processor with 2 Gb RAM; r=rigid, d=deformable, VC=virtual coupling and manipulandum transformations, StVK=deformable FEM dynamics, TT=tree traversal, TC=temporal coherence, LOD=level of detail rendered (in the particular configuration), TL=time under a tree-less pointshell traversal. All deformable models use $r = 15$.

Graphical rendering: Our pointshell objects carry two meshes, in addition to the pointshell. A *volumetric mesh* (i.e., consisting of 3D solid elements) is used for the FEM deformable object simulation. A *surface triangle mesh* is used to render the object on the screen. The volumetric mesh deformation vector u drives the deformation vector of the rendering mesh $\overline{u}$. If necessary, the rendering mesh can display higher visual detail as actually used internally for the FEM deformable object simulation. However, we do not exploit this aggressively, but instead rely on our model reduction to reduce a geometrically complex FEM simulation to a small number of deformable degrees of freedom. Each vertex of the rendering mesh is contained in some element of the simulation mesh. Using the FEM shape functions and the volumetric mesh modal matrix U, it is possible to show that $\overline{u}$ takes the form $\overline{u} = \overline{U}q$, for some precomputed *rendering mesh modal matrix* $\overline{U}$. This equation is evaluated at graphics rates in graphics hardware, to obtain the current rendering mesh deformation vector $\overline{u}$ (see Section 2.12.1). Such off-loading to the graphics card helps avoid cache pollution which would otherwise slow down the haptic cycle immediately after each graphics frame. For lighting, dynamic normals can be computed in the vertex shader just like dynamic contact normals. To render the rigid object, we use its input triangle mesh, cached in a display list.

6-DOF haptic rendering: In our first example, we simulate a detailed deformable bridge in contact with a detailed rigid dinosaur. We recorded a characteristic manipulandum trajectory, and give simulation data under this trajectory, computed offline under different force approximations (see Figure 3.27). We also demonstrate that our method can render small geometric features, by sliding the bumps on the back of the dinosaur against the bridge (see Figure 3.26). We also present a geometrically detailed deformable dragon in contact with a rigid Buddha (see Figure 3.25). We demonstrate contact between two deformable objects by simulating the deformable dragon in contact with a deformable dinosaur. Figure 3.28 demonstrates the benefits of graceful degradation to simulate very large contact areas. Example statistics are provided in Table 3.1. In all of our

examples in this thesis, haptic forces and torques are rendered stiff (i.e., near the hardware stiffness limit of our haptic device). We used the Phantom Premium 1.5 6-DOF device from Sensable Technologies (see Figure 3.25). This device can sense manipulandum positions and orientations (6-DOF input), and render both forces and torques (6-DOF output). We also ported the demo to the Phantom Premium 1.5 3-DOF device, and the Omni 3-DOF device, both from Sensable Technologies. These two devices can sense both position and orientation (6-DOF input), and can display forces, but not torques (3-DOF output). With these devices, our simulation computes both forces and torques, but renders only forces; we display torques on the screen numerically.

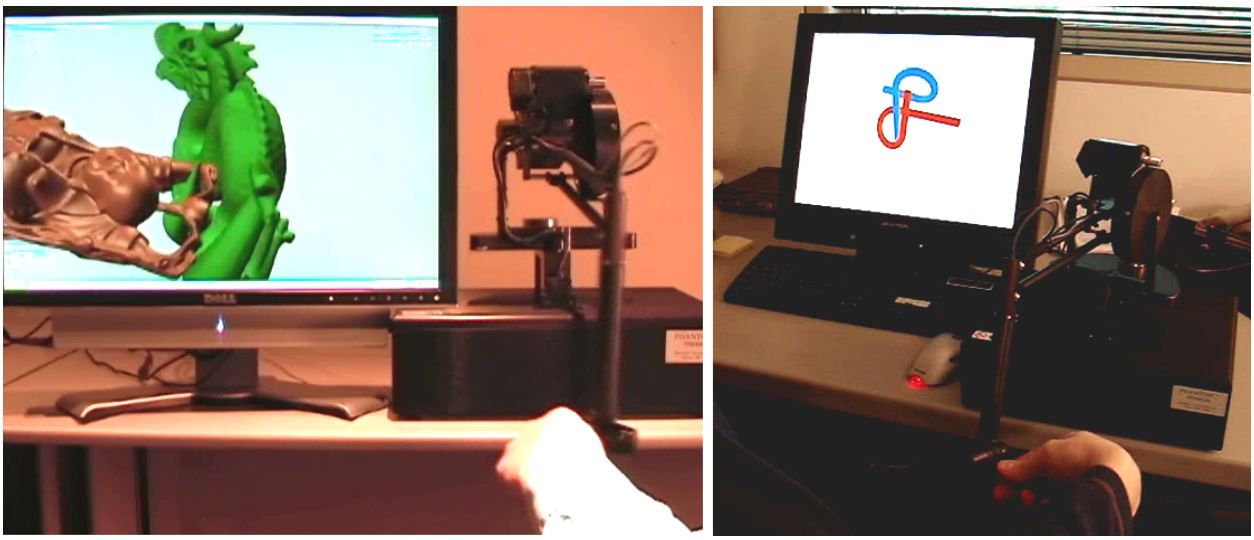


Figure 3.25: **6-DOF haptic rendering:** Left: the user is manipulating a rigid Buddha (distance field object) against the deformable dragon (pointshell object). Right: the user is solving the alpha puzzle, while the device provides contact force and torque feedback (see also Figure 3.31). The device in the images is a Phantom Premium 1.5 6-DOF device from Sensable Technologies.

Asymmetry of the contact model: Our contact model is not symmetric under a reversal of the roles of the distance field object and the pointshell object: if object A is the pointshell object, and object B is the distance field object, and A and B are in contact, then we will obtain a different contact force if A is made into the distance field object, and B is made into the pointshell object. This is immediately obvious from our contact force formula 3.1 (page 105). An intuitive explanation can also be seen from Equation 3.5 (page 108): the limit contact force is obtained by volumetrically integrating the distance field gradient over the contact volume V; if we reverse the roles, we still integrate over V, however, we must now integrate the distance field gradient *of the other object*.

We evaluated the asymmetry experimentally, by fitting a pointshell (256,000 points) and computing a signed distance field (256x256x256) to dragon geometry (150,000 triangles). We then positioned two rigid copies of the dragon in contact. In particular, we fixed the first dragon, and

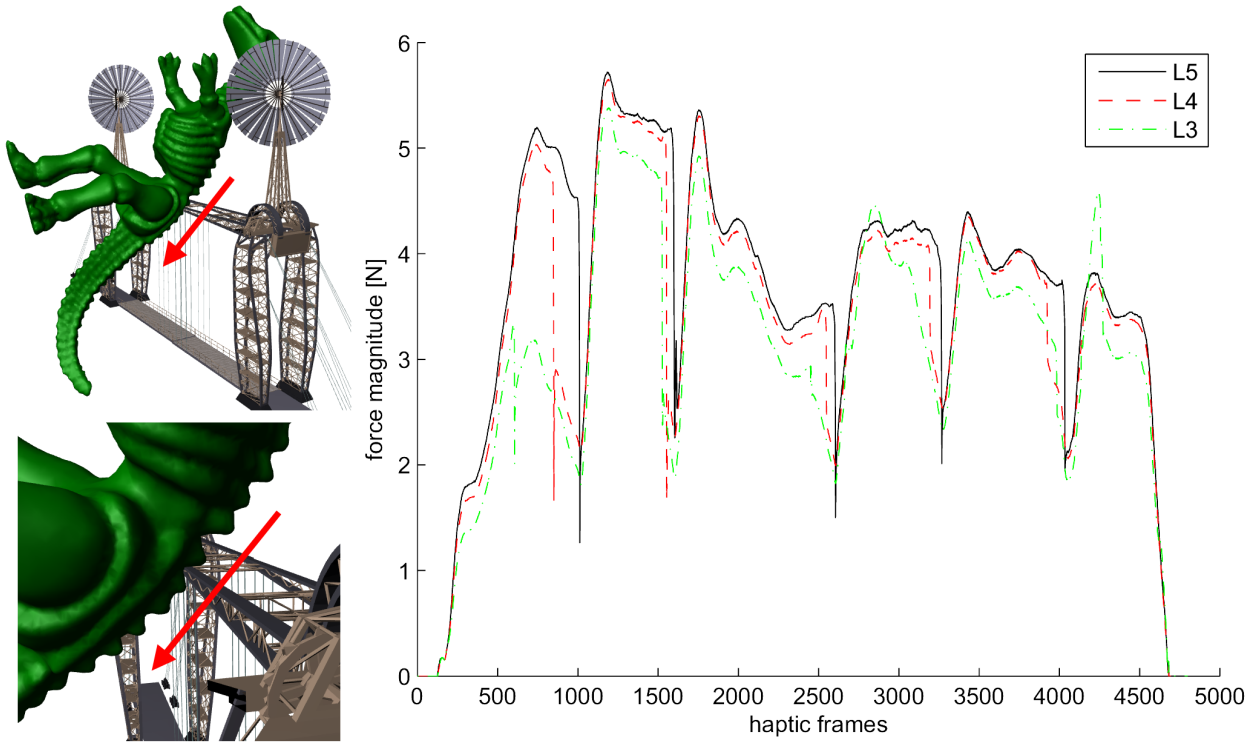


Figure 3.26: **Our method can render small features:** Left (top and bottom): resolution in our method is high enough for the user to feel the bumps on the back of the rigid dinosaur (distance field object, 256x256x256), by sliding the back of the dinosaur against the upper shelf of the bridge (pointshell object). The red arrows are indicating the direction of the dinosaur's sliding. Both objects are rigid in this experiment. Bridge pointshell has five levels of detail with 256,000 points (same L5 pointshell as in Figure 3.16, Right, on page 122). Right: the force magnitude rendered to the user, under progressively finer pointshells and a pre-recorded manipulandum trajectory. Sampling rate is 1000 frames per second. The user caused six bumps total to touch the bridge (one after another, in tangential motion). The impact of each individual bump can be clearly seen in the haptic signal, and felt in the haptic simulation.

then rigidly translated the second dragon into contact. The experiment was performed offline, with a pre-recorded linear translational trajectory for the second dragon. We repeated this experiment twice: once with the first dragon as a pointshell object and the other dragon as a distance field object, and then with the roles reversed. The trajectory was chosen such that the two dragons collide in a non-trivial distributed contact configuration (see Figure 3.29, page 145, bottom-right). We did not use virtual coupling in this experiment. Signals in this experiment are raw contact signals, obtained by evaluating our contact model for 1800 equally distributed configurations along the pre-recorded motion trajectory. For each configuration sampled, we traversed all pointshell points, and accumulated the total contact force (see Figure 3.29). The contact force was L/ℓ-scaled (see Section 3.5), and as such grows about linearly with the penetration depth.

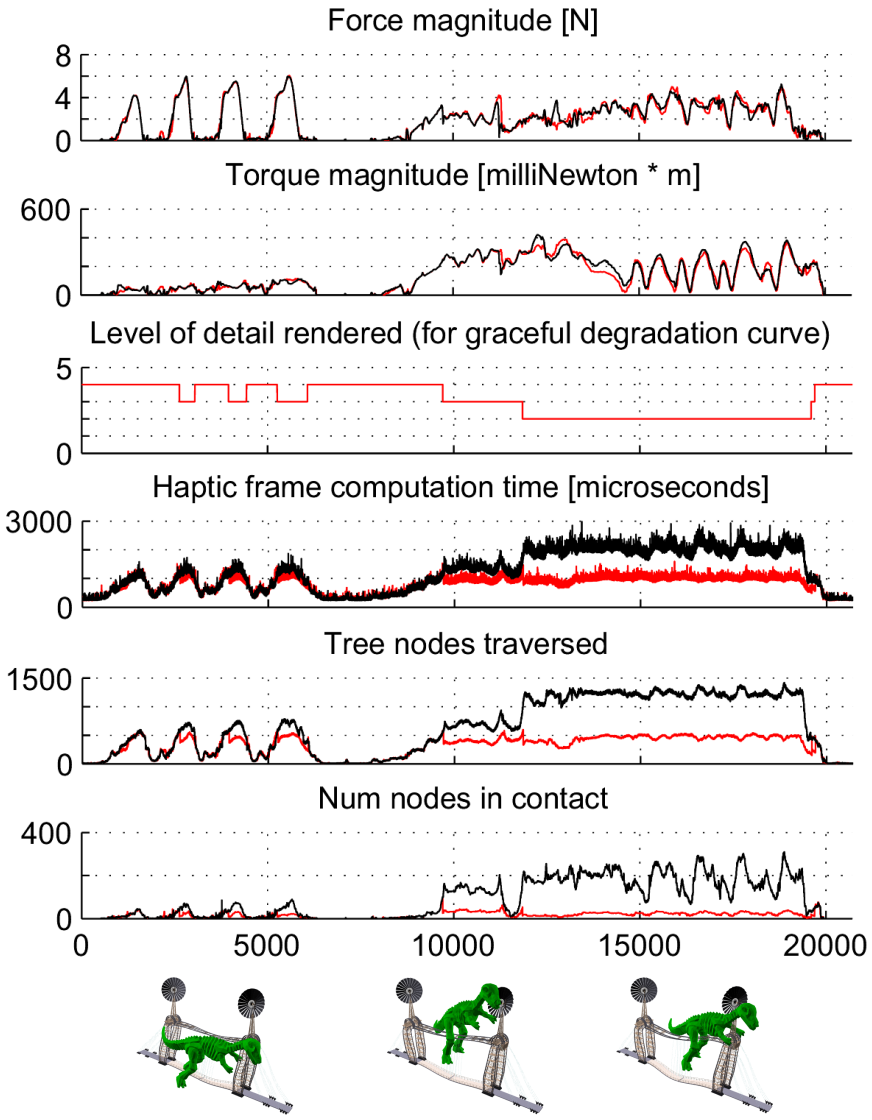


Figure 3.27: **Illustration of graceful degradation (GD):** Simulation data for a pre-recorded manipulandum trajectory is rendered in two ways: using GD (in red) with warm threshold at 600 nodes; without GD (in black) with all four tree levels. The pointshell has $85,018$ points total (same L4 pointshell as in Figure 3.15 on page 121). The common x-axis corresponds to haptic frames. This data was recorded on a three-year-old 3.0 GHz dual Intel Xeon processor with 2.75 Gb of memory, illustrating algorithm's adaptivity to different computer speeds. On the machine of Table 3.1, the black timing curve reaches a maximum of $\sim$800 μs.

The bottom chart in Figure 3.29 (page 145) gives the minimum distance field value over all pointshell points and can be understood as penetration depth. The distance field value is expressed in multiples of the signed distance field voxel size. For comparison, the diameter of the cross-

section of the largest horn on the dragon's head is about 9 distance field voxels (at its widest point). Real simulations use virtual coupling, with static damping (Section 4.8), and employ saturation (Section 4.9), which prevents the object penetrating into the other object deeper than some small threshold (typically one voxel or half a voxel in our simulations). To see the contact forces for such shallow penetration depths more clearly, we reproduced the contact forces (i.e., raw contact signal) from Figure 3.29 in greater detail (see Figure 3.30).

Application: Interactive assembly and path planning: We present an example where the user can manipulate a metallic rigid component in contact with deformable geometry typical of modern airplanes, e.g. hoses. Large hose deformations allow the operator to navigate the component in between the hoses. Another example presented is the classic "alpha" path planning puzzle [BSA00, Kuf04] (see Figure 3.31). We use the 1.0 version of the alpha puzzle (i.e., the most difficult), obtained at [Alg].

Application: Interactive gaming with haptics: Interactive haptic gaming is an emerging application area. In this example, the user can swing a detailed rigid axe to "interact" with an armored rigid knight (see Figure 1.3 on page 28). This example demonstrates crisp contact with a detailed million-point pointshell, in a rigid-only simulation. Recently, 6-DOF input devices such as Nintendo's Wii have seen a lot of commercial success, and we envision that adding appropriate force feedback could make such games even more fun.

3.14 6-DOF haptic demo

We made several of our haptic demos available on the web in the form of executables, so that other researchers can download and try them. The demos run on Sensable's force feedback devices, under Windows XP. In particular, the demos were designed for the Phantom Premium 6-DOF 1.5 device. We were also able to run them with Phantom Premium 3-DOF 1.5, and with an Omni. The demos use OpenHaptics, a haptic API from Sensable Technologies.
Our demos are at: `http://graphics.cs.cmu.edu/projects/defoContact/demo`
Note that URLs are rarely permanent due to system maintenance issues and website migrations.

3.15 Discussion

We have demonstrated that real-time haptic rendering of distributed contact between geometrically complex models is possible for both rigid and reduced deformable models. A key ingredient has been the exploitation of reduced-coordinate parameterizations of deformable models for fast dynamics, point-based deformation bounds, and output-sensitive evaluation of pointshell points and contact normals for time-critical evaluation of contact forces. We have also introduced deformable

distance fields for modal deformation models, to support time-critical contact between two deformable models. Real-time contact simulations, such as for 6-DOF haptic rendering, are now possible between rigid and reduced deformable models on commodity workstations.

There are a number of discussion items, potential limitations, and opportunities for future work. First, our deformations are not fully general; in the future, fully general complex FEM deformable objects might run at haptics rates on commodity workstations, but this is not the case today. Our deformable model is adaptive in the sense that more basis vectors can be added to the basis as processor speeds will increase. While our current demonstrations run on standard Windows or Linux platforms, one should investigate simulations on operating systems and parallel architectures that can guarantee hard real-time scheduling. While static virtual coupling greatly improves simulation stability, our algorithms come with no theoretical guarantees on passivity of our simulations (the same is true of most other notable 6-DOF haptic rendering algorithms). Developing stability and passivity proofs for 6-DOF haptic rendering is a natural area for further exploration.

The simulation can be simplified if the signed distance field is replaced by a simpler implicit function with an analytical formula, such as a sphere, or combination of infinite planes. Our offset surfaces are not reaching down to the exact geometry in case of polygon "soup" input geometry, when the contact surface is not well-defined. Quality of the offset surface degrades under offsets too small compared to distance field resolution. Large deformations can potentially introduce undersampled pointshell regions, however this issue was not problematic in our case of deformations consisting mostly of large rotations and small strain.

The deformed distance fields might be used to resolve multiple inter-scene deformable interactions. Even if deformable distance fields are not used, it would be possible to have scenes with several (smaller) deformable objects attached to a mostly rigid environment, with the rigid part carrying both a pointshell and a distance field. Each deformable object carries a pointshell that is queried against two distance fields: against the rigid part of the environment and the manipulated rigid object. If deformable objects are sufficiently far apart that they cannot touch each other, all interactions are covered.

Other reduced deformable models should be investigated, including low-resolution models that deform embedded meshes, and multi-resolution models. Algorithms to support high rank StVK models are being investigated. Although our implementation is not highly optimized, we did implement some time-critical routines (Uq and ΔNq mtx-vec multiplies, and 4x4 mtx-vec multiplications to transform points into the frame of reference of the distance field object) in assembly (SSE2 multimedia instructions), and carefully laid out data in memory to optimize cache performance; a system speedup of about $2\times$ was obtained. Haptic interaction with animated characters could also be possible by exploiting fast reduced-coordinate representations [JT05, DSP06]. Although we have focused on haptic rendering of distributed contact, deformable distance fields could also be used to support implicit rendering of single-point contact analogous to [KKSD02]. Perceptual studies may be useful in determining speed-accuracy trade-offs, such as maximum required contact resolution or perception of deformation complexity, similar to other studies done in rigid-rigid contact rendering [OL03]. Another interesting area of future work is simulating friction, which

can play an important role in applications such as analysis of mechanical part removability and accessibility.

Some applications may be limited by the fact that modification or cutting of the models is prohibited by the preprocessing employed for performance. Also, preprocessing larger models takes several hours on a single-processor computer. It would be advantageous to be able to go from input meshes to haptic simulations much faster. We have, however, demonstrated that high-quality contact signal is *possible* at haptic rates between complex deformable geometry, with a reasonable amount of precomputation. The slowest precomputation component is typically particle repulsion (not very optimized in our implementation), then deformable model reduction, and then distance field computations (we are not using the fastest algorithms available in literature, such as [SOM04, SGG+06b]). All computationally significant parts of the pipeline can be processed in parallel. Although unaddressed in our implementation, large deformations can necessitate self-collision processing, and this remains a significant computational challenge for hard real-time (haptic rendering) applications (c.f. [GKJ+05]).

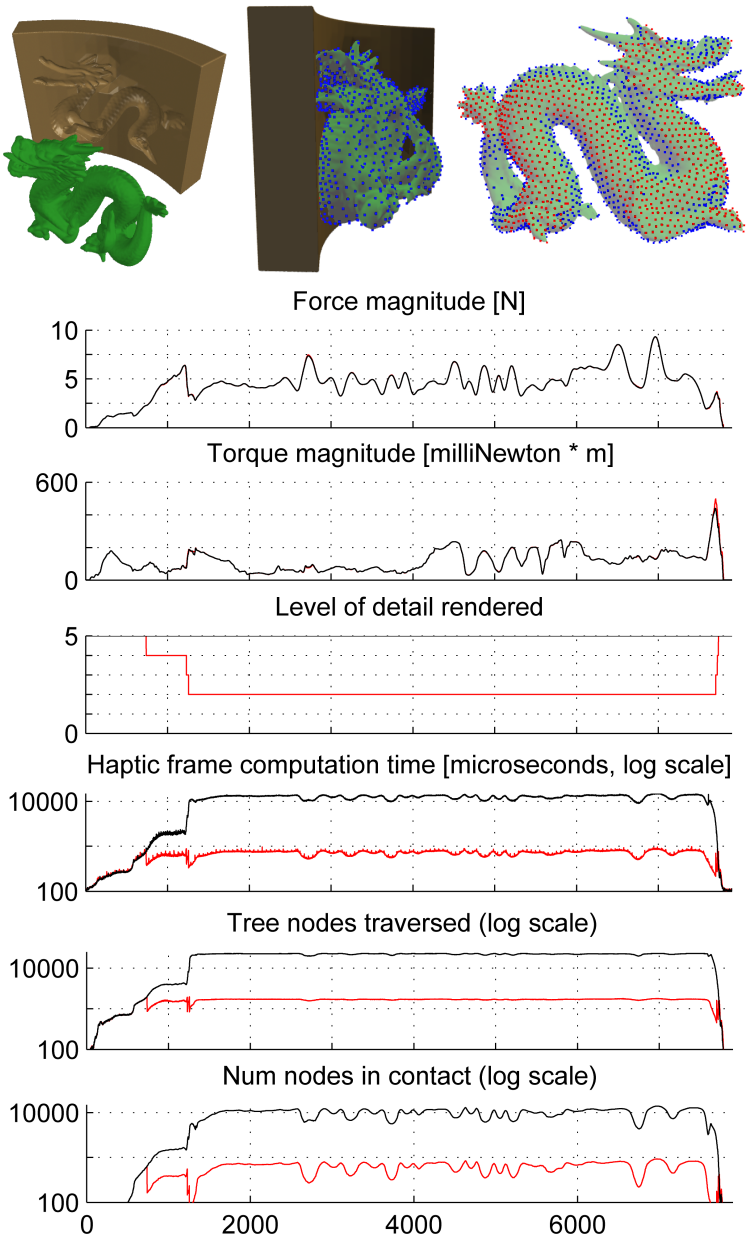


Figure 3.28: **Graceful degradation supports large conforming contacts** to allow time-critical approximation of very difficult contact configurations (here under 1 millisecond). Top Left: A dragon "peg" inserting into a matching dragon "hole" obtained via CSG difference. Top Middle: dragon manipulated haptically into the hole. Top Right: Large contact area with traversed points in blue (43% of all L2 points), and contacting points in red (21% of all L2 points). (**Bottom Graphs**) Simulation data for a pre-recorded manipulandum trajectory is rendered in two ways: using GD (in red) with warm threshold at 2000 nodes; without GD (in black) with all five tree levels. Same machine as in Table 3.1. The common x-axis corresponds to haptic frames.

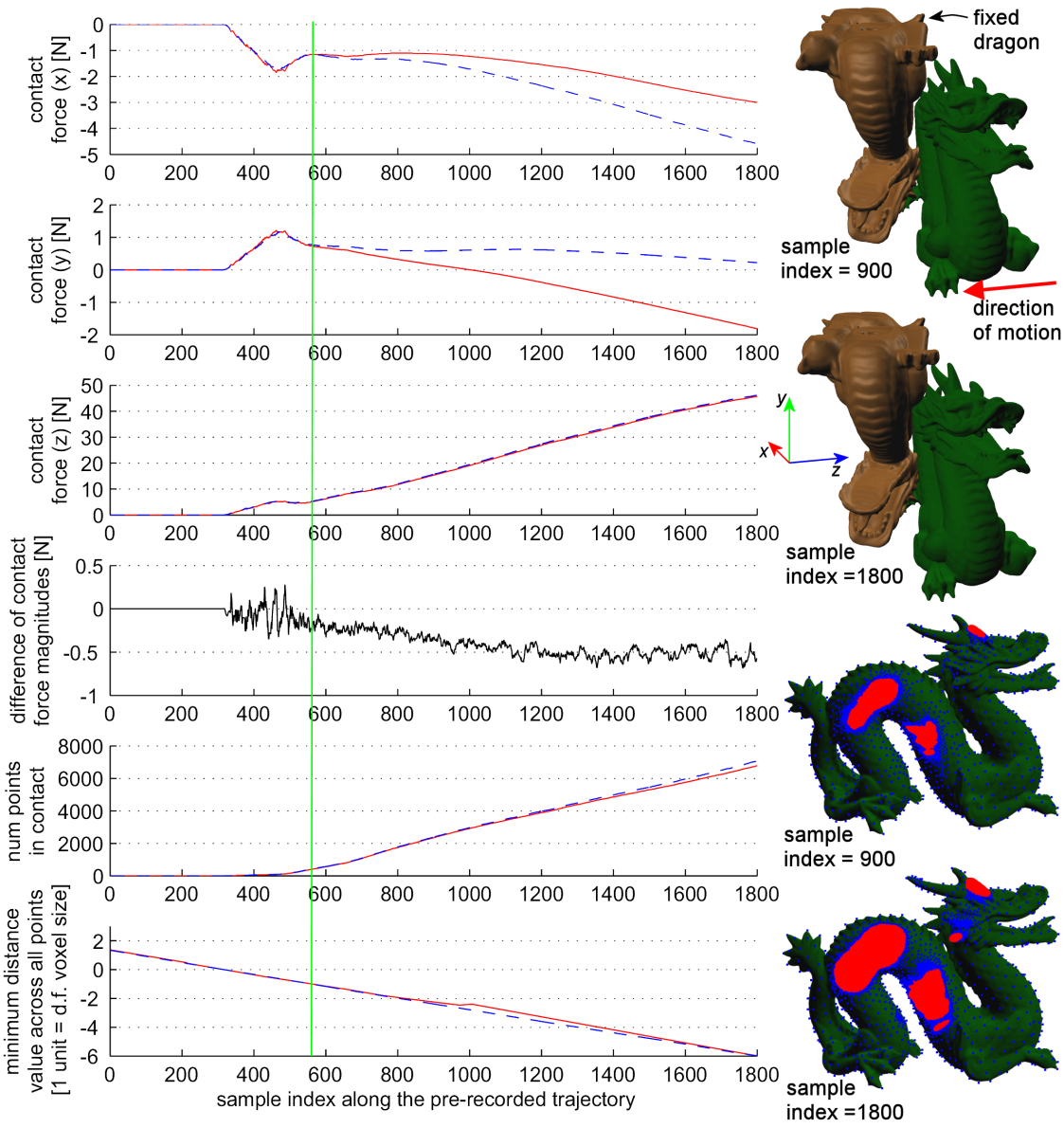


Figure 3.29: **Asymmetric contact model:** The contact force changes under role reversal, however the error is quite reasonable. The first contact case (green dragon=pointshell, brown dragon=signed distance field) is depicted with a solid red line, the reverse case (green dragon=signed distance field, brown dragon=pointshell) is shown in dashed blue. Both lines show the raw contact force on the green (i.e., moving) dragon. The top-right image shows the configurations at samples 900 and 1800, respectively. The bottom-right images depict points in contact (red=in contact, blue=traversed by tree, but not in contact). Note that the z-axis is aligned with the direction of motion, and is (about) normal to the contact areas; z-force is therefore the normal contact force and is as such substantially larger than tangential forces. Penetration depths are large in this experiment; in our simulations, virtual coupling coupling saturation would prevent penetrations deeper than about 1 voxel. The vertical green line denotes the sample index where penetration reaches 1 voxel.

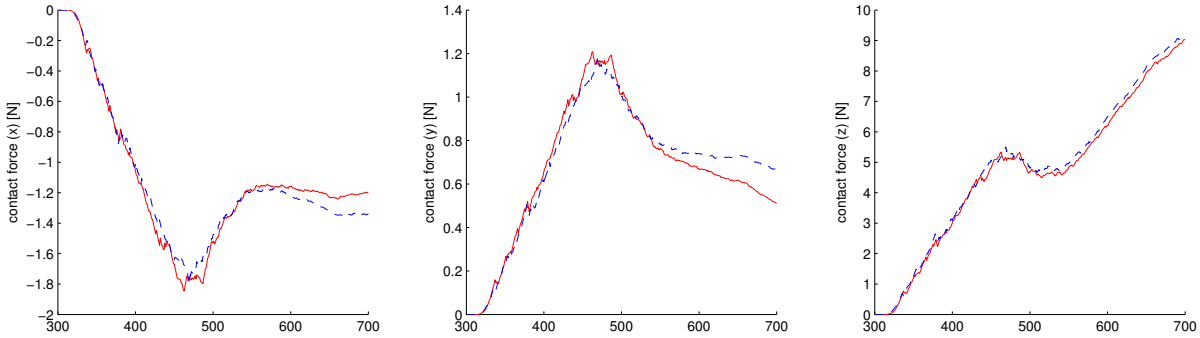


Figure 3.30: **Asymmetric contact (detailed plot for shallow penetrations):** the contact forces are not symmetric, however the difference is small for shallow penetrations. Same notation as in Figure 3.29. X-axis corresponds to sample indices.

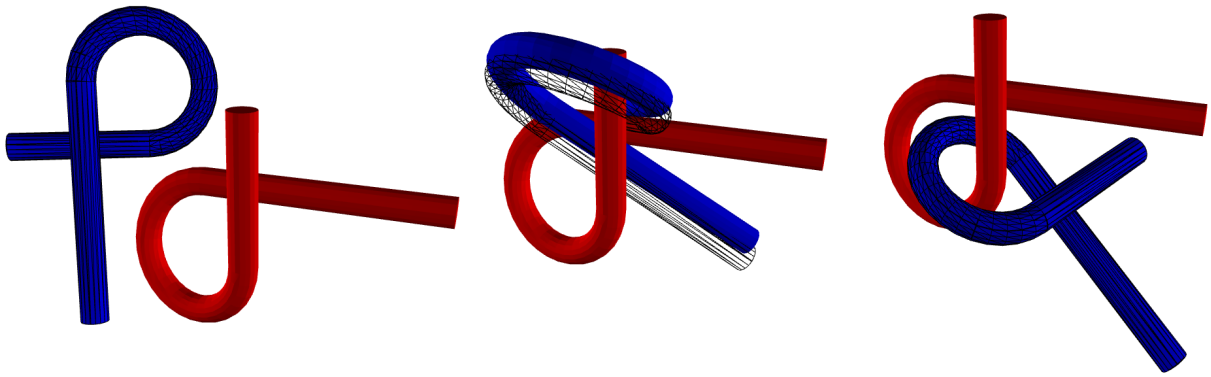


Figure 3.31: **Interactive path planning with 6-DOF haptics (Alpha puzzle):** The goal is to position the rigid blue alpha-shaped tube inside the loop of the rigid red alpha-shaped tube. The red and blue tubes have identical shapes. The red tube (pointshell object, 5571 points, 4 levels total) is fixed in space, and the blue tube (distance field, 128x128x128) is the haptic object. The gap between the two sides of each alpha (best seen with the blue alpha in the middle image) is too small for trivial attempts to succeed, both in the real-world and in our haptic simulation. With careful manipulation, the two solutions to the puzzle can be discovered. Left: initial configuration ($106\mu s$ of computation per haptic cycle, 2 tree nodes traversed, 0 points in contact). Middle: a configuration in the middle of one of the two solution paths ($156\mu s$, 233, 5). Right: puzzle solved ($108\mu s$, 16, 0). Path planning is accurate up to the penetrations permitted by maximum contact forces under virtual coupling saturation, e.g. typically one half of a distance field voxel. Also note that we could make this example deformable, but then the solution would be very easy.

Chapter 4

Haptic display of distributed contact

Until now we have abstracted 6-DOF haptic rendering as simply a matter of computing contact forces and torques at high rates. However, as described in Section 3.1.1, the penalty forces and the corresponding torques cannot simply be sent to the haptic device: each point in contact adds to the overall stiffness of the system and if enough points are added, maximum renderable stiffness of the haptic device is exceeded easily, resulting in device instabilities. Instead, it is customary to separate the *simulation position* of the haptic object from the position imposed by the haptic manipulandum (*manipulandum position*), and connect the two with *virtual coupling* [CMJ95]. We introduced virtual coupling in Section 3.1.1; we reiterate key points for chapter completeness. Common choices for virtual coupling are a mass-spring-damper (dynamic virtual coupling), or a spring only (static virtual coupling). Virtual coupling tries to align the simulation position and orientation to the manipulandum position and orientation (see Figure 3.5 on page 99, or also Figure 4.1 for an illustration). Virtual coupling consists of two separate 3D mass-spring-dampers (or springs): one for position and one for orientation. As a starting point, we adopt *static* virtual coupling of [WM03] since it needs a relatively small number of parameters, and extend it by introducing *static damping*. We give explicit formulas for contact forces and torques, their gradients, as well as virtual coupling forces and torques, and their gradients. We also include large rotation terms in the static equilibrium equations. Another contribution is our handling of rank-deficient virtual coupling force and torque gradients that can occur during virtual coupling saturation.

The manipulandum and the simulation object: One good way to visualize virtual coupling is to imagine two coordinate systems, one for the simulation copy and one for the manipulandum copy of the haptic object. We will call these two copies the "simulation object," and "manipulandum," respectively. Both coordinate systems are attached to some material location X_{grasp} on the object; imagine they are rigidly tied to a (small interior) piece of the object. The location X_{grasp} is same for both haptic object copies. The two coordinate systems transform rigidly with their respective haptic object copy. Denote the current world-coordinate location of the grasping point on the simulation object and manipulandum by $x_{\mathrm{grasp,s}}$ and $x_{\mathrm{grasp,m}}$, respectively. Both of these locations are functions of time, as the user moves the manipulandum and as the simulation

updates the position (and orientation) of the simulation object. Virtual coupling tries to align the two coordinate systems: the positional virtual coupling tries to align the origins and the rotational virtual coupling tries to align the coordinate axes. The exact formulas for these forces and torques are given in Section 4.3. During contact, the two coordinates systems separate, as the user moves the manipulandum into contact, and the contact forces keep the simulation object away from contact. The virtual coupling forces and torques, as felt by the manipulandum copy, are the forces and torques that are rendered to the user. The user feels as if they are holding the manipulandum and are connected to the simulation object via a spring.

Choice of grasping location: The particular location X_{grasp} can be chosen anywhere inside the object (or even outside), and can be interpreted as grasping the haptic object at a particular location on the object. The grasping location could be easily modified during the simulation if necessary. However, everywhere in this thesis we will assume that the grasping location does not change during the simulation. The virtual coupling forces and torques experienced during the simulation will of course differ depending on the choice of the grasping location. For example, if one grabs a long pole at one end, stronger torques will be felt when the other end of the pole enters contact with some stationary rigid object than if the pole were grabbed near its center of mass (see Figure 4.1).

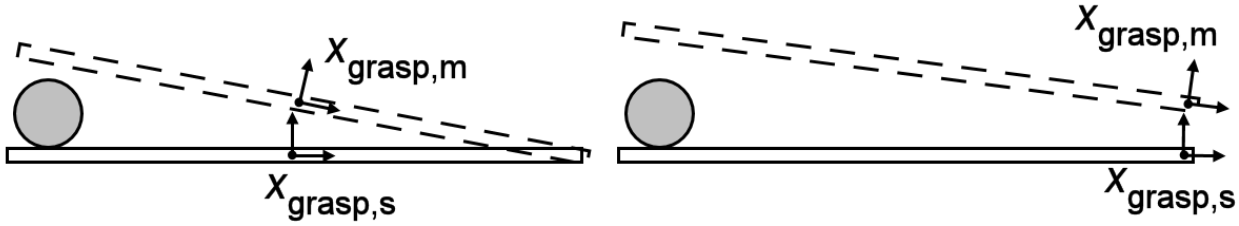


Figure 4.1: **Grasping location affects virtual coupling forces and torques:** The user is manipulating this long pole against a rigid obstacle (a cylinder viewed from top), and will feel different forces and torques depending on the choice of X_{grasp} (Left: at pole center, Right: at pole far end). Stippled line denotes the manipulandum and solid line denotes the simulation object. Virtual coupling tries to align the two coordinate systems.

The structure of one haptic cycle: In each haptic cycle, we first read the position and orientation of the haptic device manipulandum. We then traverse the nested point-tree to compute contact penalty forces and torques, and their gradients with respect to the *simulation* position of the haptic object (see Figure 3.6 on page 106). Next, we compute the virtual coupling force and torque, and their gradients. The net force and torque on the simulation copy of the haptic object under a (small)

incremental displacement Δx and (small) rotation $\Delta\omega \in \mathbb{R}^3$ of the simulation object, are then

$$F_{\text{net}} = F_{\text{VC}} + F_{\text{C}} + \Big(\frac{\partial F_{\text{VC}}}{\partial x} + \frac{\partial F_{\text{C}}}{\partial x}\Big)\Delta x + \Big(\frac{\partial F_{\text{VC}}}{\partial \omega} + \frac{\partial F_{\text{C}}}{\partial \omega}\Big)\Delta\omega, \tag{4.1}$$

$$T_{\text{net}} = T_{\text{VC}} + T_{\text{C}} + \Big(\frac{\partial T_{\text{VC}}}{\partial x} + \frac{\partial T_{\text{C}}}{\partial x}\Big)\Delta x + \Big(\frac{\partial T_{\text{VC}}}{\partial \omega} + \frac{\partial T_{\text{C}}}{\partial \omega}\Big)\Delta\omega, \tag{4.2}$$

where $F_{\text{VC}}, T_{\text{VC}}$ denote current virtual coupling force and torque on the simulation object, and $F_{\text{C}}, T_{\text{C}}$ denote the sum of all contact forces and torques. We displace the simulation object such that the net force and torque on the object vanish under this first-order model ($F_{\text{net}} = T_{\text{net}} = 0$), by solving the 6×6 linear system of Equations 4.1, 4.2 for $(\Delta x, \Delta\omega)$. This system is (in general) not symmetric and we use a LU decomposition to solve it (but see also Section 4.9). Solution is fast, typically ~ 5 microseconds on our hardware. Finally, we compute the virtual coupling force and torque with respect to the new simulation position, and render them to the user, with a negative sign (Newton's 3rd law: virtual coupling force/torque on manipulandum is opposite to virtual coupling force/torque on simulation object). See Figure 4.2 for a schematic block diagram of one haptic cycle. In the next sections, we describe how we compute the virtual coupling and contact forces, torques, and their derivatives.

How simulation update rate affects static virtual coupling stability: The static virtual coupling procedure can be seen as a Newton-Raphson solver which tries to position the simulation object into equilibrium under the virtual coupling and contact forces and torques. The solver is aiming at a moving target, as the manipulandum position and orientation constantly change. Each haptic cycle is one solver iteration: traversing the nested point-tree and evaluating the virtual coupling model is equivalent to computing the current function value $f(x)$ and its derivative $f'(x)$. Solving Equations 4.1 and 4.2 for $F_{\text{net}} = T_{\text{net}} = 0$ is equivalent to solving $f(x) + f'(x)\Delta x = 0$. Higher update rates allow f and f' to be sampled more often in time. Hence, the Newton-Raphson solver is closer to the equilibrium solution at the beginning of each haptic cycle, and therefore convergence is better. It is possible for static virtual coupling to not converge. We found our simulations to be very stable, even at stiff settings, however, they are not perfect: in case of convergence problems, static virtual coupling tends to get stuck in a limit cycle, which manifests as haptic signal noise. One solution in such cases would be to momentarily reduce the maximum allowable simulation object displacements and orientations (see Section 4.6), until convergence is restored.

Stability is related to the ratio between update rate and manipulandum velocities: Static virtual coupling does not incorporate dynamics and hence does not know about time explicitly. Let us disregard issues related to deformable object simulation and the fact that manipulandum is held by a human (i.e. assume the manipulandum is driven according to a pre-specified trajectory by a stiff robot). Then, a simulation running at 1000Hz will give equally stable forces as a simulation running at 5000Hz with 5x higher average manipulandum velocities, because a 5000Hz simulation will have sampled five times more manipulandum samples. For a fixed average manipulandum

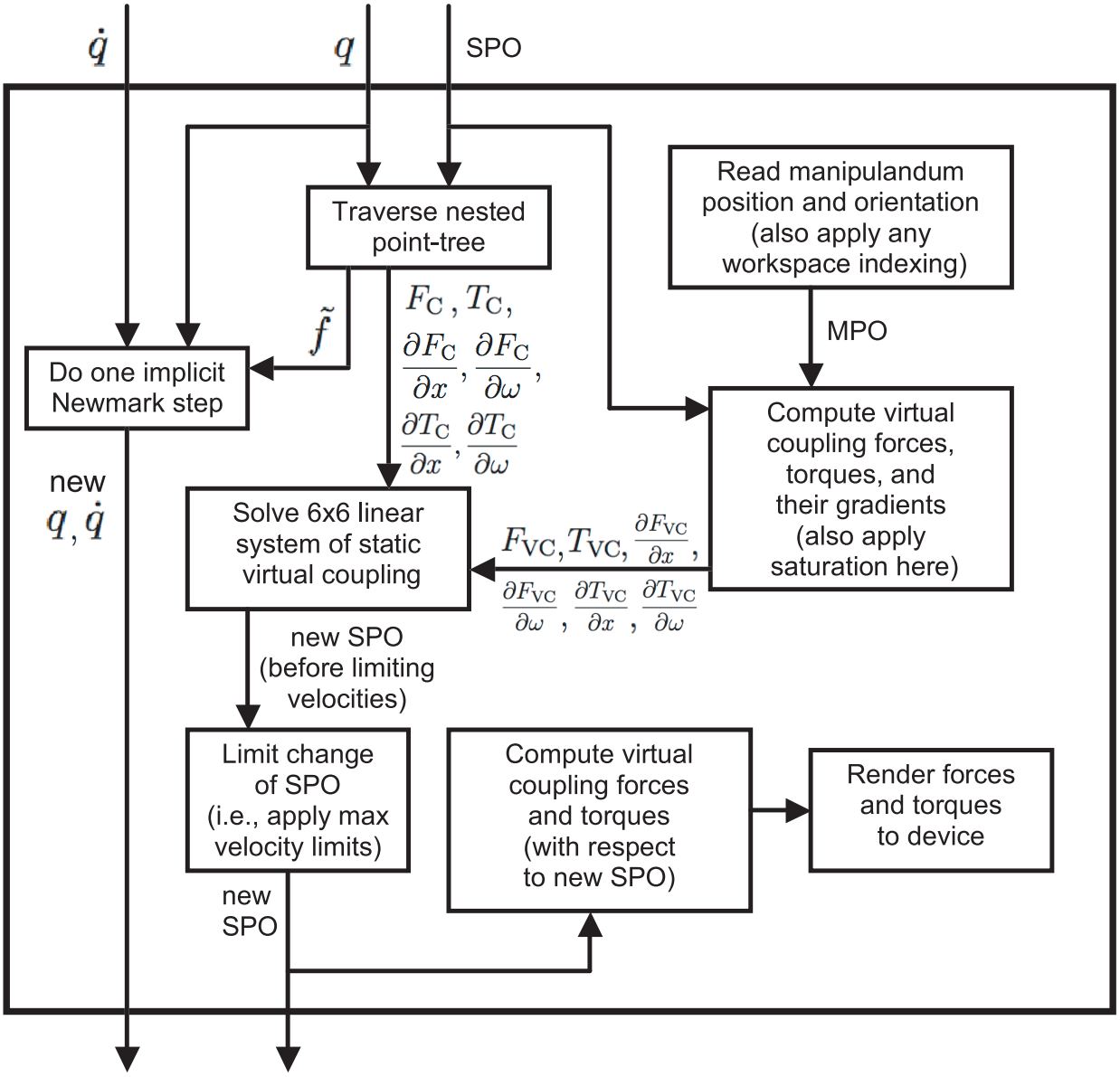


Figure 4.2: **Block diagram of one haptic cycle:** SPO=Simulation object Position and Orientation, MPO=Manipulandum Position and Orientation.

velocity, higher update rates imply that the manipulandum moves a shorter distance within each haptic cycle, which is beneficent to convergence of static virtual coupling.

4.1 The contact force and torque

Without loss of generality, we assume (everywhere in this thesis) that the pointshell object is anchored (i.e., it does not possess rigid degrees of freedom, but can undergo reduced deformations), and that the user is manipulating the distance field object; that is, the distance field object is the haptic object. As described in Section 3.5, we use penalty contact forces. The points of the pointshell are transformed into the frame of reference of the distance field object (we coded the required 4×4 mtx-vec multiplications using Intel's SSE2 assembly language multimedia instructions), then looked up into the signed distance field to determine the penetration depth d.

Contact force: Our contact force, arising from contacting point i and acting on the simulation object, is $F_{C,i} = k_C d_i N_i$ (see Figure 4.3, and also Figure 3.6 on page 106; note that $d_i < 0$ due to contact). Here, N_i is the inward normal of point i, that is, a normal pointing in the interior of the pointshell object (and typically away from the distance field object). The contact force $F_{C,i}$ and the normal N_i are expressed in the world-coordinate system. Note that this contact model obtains the penetration depth from one object and contact normal from the other object. This construction avoids using the distance field gradients for contact force directions, as these gradients are discontinuous along the object's (interior) medial axes. The total contact force on the simulation object is obtained by summing the contributions of all points: $F_C = \sum_i F_{C,i}$.

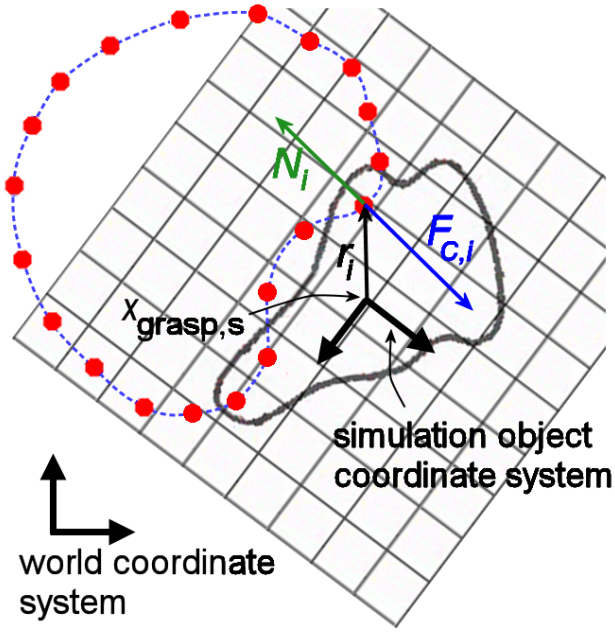


Figure 4.3: **The contact model**

Contact torque: The torque on the simulation copy of the distance field object, originating from point i, is $r_i \times F_{C,i}$, where r_i is the vector from $x_{\text{grasp,s}}$ (expressed in the world coordinate system)

to the world-coordinate location of point i on the pointshell object (see Figure 4.3).

4.2 The contact force and torque gradients

During every haptic cycle, it is necessary to compute the gradients

$$\frac{\partial F_{\mathrm{C}}}{\partial x}, \quad \frac{\partial F_{\mathrm{C}}}{\partial \omega}, \quad \frac{\partial T_{\mathrm{C}}}{\partial x}, \quad \frac{\partial T_{\mathrm{C}}}{\partial \omega}, \tag{4.3}$$

so that we can then solve the Equations 4.1 and 4.2. Here, $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial \omega}$ denote the gradient with respect to a small position and small orientation change of the simulation object, respectively. That is, we need to determine how the contact force and torque, on the distance field object, change if the distance field object displaces a small (infinitesimal) amount of Δx, and/or rotates by a small (infinitesimal) rotation $\Delta\omega$. These changes can be visualized as a change in the position and orientation of the coordinate system attached to the simulation object. The gradients are a limit under infinitesimal changes, therefore we are interested in small rotations and can approximate the rotation with an infinitesimal rotation. Every point in contact contributes to these gradients. The total gradient is obtained by summing the contributions from all points in contact. In the following, we describe how we compute the contact gradient contribution for each individual point in contact.

For writing simplicity, we will omit the point index i for the rest of this section. Consider a pointshell point in contact, and let r be the world-coordinate vector from $x_{\mathrm{grasp,s}}$ to the world-coordinate location of the point. The contact force on the simulation object is $F = k_C \cdot d(x_{\mathrm{grasp,s}} + r) \cdot N_0$, where N_0 is the point's inward normal, expressed in the world coordinate system. The quantity $d(\_)$ denotes the signed distance field value, in this case at the location corresponding to $x_{\mathrm{grasp,s}} + r$. The change in the contact force and torque due to displacing the distance field object by $(\Delta x, \Delta\omega)$ is identical to the change incurred if the distance field object stayed fixed, but the point is displaced according to $(-\Delta x, -\Delta\omega)$.[1] That is, the point's position changes to $p(\Delta x, \Delta\omega) = x_{\mathrm{grasp,s}} + r - \Delta x - (\Delta\omega) \times r$, and the point's normal changes from N_0 to $N(\Delta x, \Delta\omega) = N_0 - (\Delta\omega) \times N_0$. Therefore, we have

$$\frac{\partial p}{\partial x} = -I, \qquad \frac{\partial p}{\partial \omega} = \tilde{r}, \qquad \frac{\partial N}{\partial x} = 0, \qquad \frac{\partial N}{\partial \omega} = \widetilde{N_0}. \tag{4.4}$$

Here, $\tilde{x}$ denotes the skew symmetric matrix ($\tilde{x}^T = -\tilde{x}$) corresponding to the cross product with $x \in \mathbb{R}^3$, i.e., $\tilde{x} \cdot y = x \times y$ for all $y \in \mathbb{R}^3$. All the gradients of scalar quantities are assumed to be laid out in the 1×3 row vector format. All the gradients of vector quantities are 3×3 matrices, where each row gives the gradient of the corresponding component of the vector. Using the chain

[1] We also derived direct formulas using $(\Delta x, \Delta\omega)$, however the simulation performed worse than if using the mirror $(-\Delta x, -\Delta\omega)$ construction described in this section.

rule and the multiplication rule, it follows that

$$\frac{\partial F}{\partial x} = kN_0 \otimes \Big(\frac{\partial d}{\partial p}\frac{\partial p}{\partial x}\Big)^T + kd\frac{\partial N}{\partial x} = -kN_0 \otimes \Big(\frac{\partial d}{\partial p}\Big)^T, \tag{4.5}$$

$$\frac{\partial F}{\partial \omega} = kN_0 \otimes \Big(\frac{\partial d}{\partial p}\frac{\partial p}{\partial \omega}\Big)^T + kd\frac{\partial N}{\partial \omega} = -kN_0 \otimes \Big(\tilde{r}\Big(\frac{\partial d}{\partial p}\Big)^T\Big) + kd\widetilde{N_0}, \tag{4.6}$$

$$\frac{\partial T}{\partial x} = \tilde{F} - \tilde{r}\frac{\partial F}{\partial x} \tag{4.7}$$

$$\frac{\partial T}{\partial \omega} = -\tilde{F}\tilde{r} - \tilde{r}\frac{\partial F}{\partial \omega}. \tag{4.8}$$

Tensor product $a \otimes b$ of vectors $a, b \in \mathbb{R}^3$ denotes the matrix ab^T. Also, we have used the rule that the gradient of a cross product $u \times v$, where both u and v are vector functions of x, is

$$\frac{\partial}{\partial x}\Big(u(x) \times v(x)\Big) = -\widetilde{v(x)}\frac{\partial u}{\partial x} + \widetilde{u(x)}\frac{\partial v}{\partial x}. \tag{4.9}$$

Evaluation of the Equations 4.5-4.8 requires the computation of $\frac{\partial d}{\partial p}$, the gradient of the distance field with respect to the query location. This gradient could be obtained by differentiating analytically the trilinear interpolation functions, as explained in Section 3.6.3. Also note that the gradient $\frac{\partial d}{\partial p}$ must be evaluated with respect to the rotated signed distance field:

$$\frac{\partial d}{\partial p} = \frac{\partial d}{\partial X}\frac{\partial X}{\partial p} = \frac{\partial d}{\partial X}R^T. \tag{4.10}$$

Here, R denotes the rotation from the material space of the distance field object to the current configuration of the distance field object (we have $p(X) = x_{\text{grasp,s}} + R \cdot X$), and $\frac{\partial d}{\partial X}$ denotes the gradient in the rest configuration; i.e., the configuration in which the distance field was precomputed. Distance field gradient computation is only necessary for points in contact, not all pointshell points. Even if the contact gradient vanishes due to crossing an interior medial axis, the degenerate rank handling technique of Section 4.10 takes care of the singularity.

In our implementations, however, we make the assumption that the distance field gradient is oriented in the direction of N_0, i.e., $\frac{\partial d}{\partial p} = N_0$. This approximation avoids the exact gradient computation, and is motivated by the fact that during contact, the point's normal will typically be orthogonal to the penetrated surface, hence approximately colinear with the distance field gradient. Under this assumption, our contact force and torque gradients become

$$\frac{\partial F}{\partial x} = -kN_0 \otimes N_0, \qquad \frac{\partial F}{\partial \omega} = k\Big(N_0 \otimes N_0\Big)\tilde{r} + kd\tilde{N}_0, \tag{4.11}$$

$$\frac{\partial T}{\partial x} = \tilde{F} - k\tilde{r}N_0 \otimes N_0, \qquad \frac{\partial T}{\partial \omega} = -\tilde{F}\tilde{r} + k\tilde{r}\Big(N_0 \otimes N_0\Big)\tilde{r} + kd\tilde{r}\tilde{N}_0. \tag{4.12}$$

Finally, we note that this gradient computation scheme assumes that the pointshell does not undergo any deformations in between two haptic cycles. It would be possible to add partial derivatives of the contact force and torque with respect to q, i.e., the reduced coordinates of the current

haptic cycle. However, we would then need to solve a $(6 + r) \times (6 + r)$ linear system instead of the 6×6 system of Equations 4.1 and 4.2, and the $r \times r$ system inside the implicit Newmark integrator. This $(6 + r) \times (6 + r)$ system would incorporate the implicit Newmark dynamics, and would thus simultaneously update the position and orientation of the distance field object, and the reduced coordinates of the pointshell object. We did not implement this extension due to a significantly larger implementation complexity, and to some extent, also slower simulation times: a 6×6 system solve plus $r \times r$ system solve is faster than a $(6 + r) \times (6 + r)$ system solve, especially for values of r typically used in our simulations (e.g., $r = 15$). The speed difference in the system solves, however, becomes marginal with larger values of r. Adding such contact force gradients with respect to the reduced coordinates is a potential area for future work.

4.3 The virtual coupling force and torque

The virtual coupling force on the simulation object is

$$F_{\mathrm{VC}} = k_{VC}(x_{\mathrm{grasp,m}} - x_{\mathrm{grasp,s}}), \tag{4.13}$$

where k_{VC} is the *virtual coupling stiffness* coefficient (units are N/m). This linear relationship between displacement and virtual coupling force is, however, only used for small displacements. For large displacements, the force is designed to saturate to a constant maximum value, to prevent deep penetrations and potential pop through (discussed in Section 4.9).

The virtual coupling torque T_{VC} tries to align the orientation of the simulation object to the manipulandum. For this purpose, one can conceptually translate the two coordinate systems of the two copies of the haptic object such that they have a common origin. Then, there exists a unique rotation that would transform the simulation coordinate system to the manipulandum coordinate system. Every rotation is a rotation around some axis for some angle $\theta \in [0, \pi]$. This axis is unique for $\theta \in [0, \pi)$ (there are two opposite axis choices when $\theta = \pi$) and is called the *equivalent axis of rotation*. The virtual coupling torque is set to be oriented in the direction of the equivalent axis of rotation. Its magnitude in our implementation equals $k_{VC,\mathrm{torque}}\sin(\theta/2)$, which means that the torque saturates to a plateau as $\theta \rightarrow \pi$. Here, $k_{VC,\mathrm{torque}}$ denotes the *rotational virtual coupling stiffness* (units are Nm). Its value is independent from k_{VC}. If k_{VC} is kept constant, but $k_{VC,\mathrm{torque}}$ is decreased, the virtual coupling torques become progressively weaker than the virtual coupling forces, and object's rotation becomes progressively more compliant. Under such circumstances, the object tends to change orientation easily to avoid contact. Conversely, if one increases $k_{VC,\mathrm{torque}}$, rotations become stiffer, allowing smaller rotation discrepancies between the manipulandum and simulation object. Increasing $k_{VC,\mathrm{torque}}$ too much causes deep penetrations and haptic instabilities. If the user orients the manipulandum deep into contact, the simulation object has to follow deep into contact due to high rotational virtual coupling stiffness. This leads to the same problems as if simulation used no virtual coupling (direct rendering).

We use unit quaternions to represent haptic object orientations. Denote the quaternions corresponding to current rotations of the manipulandum and simulation object as $q_{\mathrm{m}}, q_{\mathrm{s}}$, respectively.

The question is now, how to express our definition of T_{VC} from the previous paragraph in the language of quaternions. As is well known, opposite quaternions q and $-q$ represent the same rotation [Sho85, Par01], and therefore our formulation will have to be invariant to multiplying either of q_{m} or q_{s} (or both) by -1. After some algebra, it can be shown that

$$T_{\mathrm{VC}} = k_{VC,\mathrm{torque}}\mathrm{vector}\Big(\mathrm{toPositiveHemisphere}\Big(q_{\mathrm{m}} \cdot q_{\mathrm{s}}^{-1}\Big)\Big). \tag{4.14}$$

Here, $\mathrm{vector}(q)$ is the vector part of the quaternion, that is, if $q = s + xi + yj + zk$, we have $\mathrm{vector}(q) = (x, y, z)$. The function $\mathrm{toPositiveHemisphere}(q)$ moves the unit quaternion q to the positive unit quaternion hemisphere, that is, it establishes $s \geq 0$:

$$\mathrm{toPositiveHemisphere}(q) = \begin{cases} q\,; & \text{if } s \geq 0, \\ -q\,; & \text{if } s < 0. \end{cases} \tag{4.15}$$

Therefore, our particular choice for T_{VC} (involving $\sin(\theta/2)$) is convenient because it leads to a simple quaternion expression for T_{VC}; we use expression 4.14 in our implementation.

The function $\mathrm{toPositiveHemisphere}(q)$ is necessary to prevent the virtual coupling torque from pointing in the wrong direction, as described next. For a unit quaternion $q = s + xi + yj + zk$, the corresponding rotation is encoded as

$$q = \cos(\theta/2) + \sin(\theta/2)(a_x i + a_y j + a_z k), \tag{4.16}$$

where (a_x, a_y, a_z) is a unit rotation axis, and θ is a rotation angle in radians. Given a particular quaternion q, quantities θ and (a_x, a_y, a_z) in Equation 4.16 are unique, if one assumes $\theta \in [0, 2\pi]$ and $a_x^2 + a_y^2 + a_z^2 = 1$. Note that we must include the $\theta = 2\pi$ case; it is necessary for representing the identity rotation of $q = -1$. The unique quantities change if q is multiplied by -1, even though the underlying rotation does not. For quaternions with $s \geq 0$, the unique angle θ lies on $[0, \pi]$, whereas for $s < 0$, it lies on $(\pi, 2\pi]$. Without using $\mathrm{toPositiveHemisphere}(q)$, if $q = q_{\mathrm{m}}q_{\mathrm{s}}^{-1}$ happens to have the property $s < 0$, the quaternion q will want to rotate the simulation object by an angle $\theta \in (\pi, 2\pi]$ around (a_x, a_y, a_z). In other words, this is actually a rotation by an angle of $2\pi - \theta \in [0, \pi)$ around $-(a_x, a_y, a_z)$, so the torque actually needs to act along the axis $-(a_x, a_y, a_z)$. However, we have $\sin(\theta/2) \geq 0$, which makes $\mathrm{vector}(q) = \sin(\theta/2)(a_x i + a_y j + a_z k)$, and therefore torque, point in the wrong opposite direction! This can lead to a difficult implementation bug. If one applies $\mathrm{toPositiveHemisphere}(q)$ before applying $\mathrm{vector}(\_)$, one ensures that one is always in the $\theta \in [0, \pi]$ case, making the torque in Equation 4.14 always point in the correct direction.

Our definition of T_{VC} allows us to compute the torque without ever explicitly constructing θ or (a_x, a_y, a_z), avoiding the need for inverse trigonometric functions. The same property also holds with the virtual torque gradients, which we will derive in the next Section 4.4. However, in some cases it might be advantageous to use other monotonically increasing functions of θ for torque magnitude (as opposed to $\sin(\theta/2)$). For example, if one wanted torques to saturate at some angle $\alpha \in (0, \pi]$, one could use $\sin(\theta/2)/\sin(\alpha/2)$ for $\theta \leq \alpha$, and 1 otherwise. This technique could be

useful in preventing the user from applying large torques (which could overpower penalty contact forces), i.e., it could be used for torque saturation, as described in Section 4.9. These alternative torque functions would require an explicit computation of θ and (a_x, a_y, a_z) (which would be cheap; on the order of 1 microsecond or less). Again, discussion from the previous paragraph should be followed to orient the axis correctly.

4.4 The virtual coupling force and torque gradients

In this section, we discuss virtual coupling force and torque gradients with respect to the position and orientation of the simulation object, assuming that the manipulandum copy is held fixed. Such an assumption is reasonable given that the manipulandum position and orientation is only sampled at the beginning of the haptic cycle, and their subsequent trajectory during the haptic cycle is not known. The virtual coupling force depends only on the difference in current world-coordinate locations of grasping points $x_{\text{grasp,s}}$ and $x_{\text{grasp,m}}$ (Equation 4.13). Also, virtual coupling torque does not change under translations of the simulation object. Hence, we have

$$\frac{\partial F_{\text{VC}}}{\partial x} = -I, \qquad \frac{\partial F_{\text{VC}}}{\partial \omega} = 0, \qquad \frac{\partial T_{\text{VC}}}{\partial x} = 0. \tag{4.17}$$

The trickiest gradient to compute is the torque gradient with respect to the orientation. First of all, the virtual coupling torque has a singularity in the extreme case where the manipulandum and virtual object orientation differ by a rotation of 180 degrees. In this case, there are two equally plausible opposite choices for the direction of the virtual coupling torque, each trying to unrotate the simulation object in one of the two ways toward the orientation of the manipulandum. The particular chosen torque direction is not very significant in practice because both directions are physically plausible and because such extreme rotations are rarely encountered. Except for this singularity, the gradient of the virtual coupling torque of Equation 4.14, with respect to a small incremental simulation object rotation, is a well-defined and smooth function of the orientation difference quaternion $q_{\text{m}}q_{\text{s}}^{-1}$. Here is how it can be derived.

Assume that the current simulation object orientation quaternion q_s has been chosen such that $\text{toPositiveHemisphere}(q_{\text{m}}q_{\text{s}}^{-1}) = q_{\text{m}}q_{\text{s}}^{-1}$; if not, multiply q_s by -1. This potential sign flipping does not affect the virtual torque (and therefore neither its gradient), as explained in Section 4.3. The singular 180 degree configuration corresponds to the case where the scalar part of $q_m q_s^{-1}$ is zero. Since we are assuming a non-singular configuration, it follows that $q_m q_s^{-1}$ will remain on the positive hemisphere even under small perturbations of q_s. Therefore, we can safely simplify Equation 4.14 to

$$T_{\text{VC}} = k_{VC,\text{torque}} \,\text{vector}\Big(q_{\text{m}}q_{\text{s}}^{-1}\Big) = -k_{VC,\text{torque}} \,\text{vector}\Big(q_{\text{s}}q_{\text{m}}^{-1}\Big). \tag{4.18}$$

For the purpose of deriving the formula for $\frac{\partial T_{\text{VC}}}{\partial \omega}$, consider (an imaginary) process where the simulation object starts at the rotation corresponding to q_{s}, and then rotates with some (arbitrary, but

constant in time) world-coordinate angular velocity $W \in \mathbb{R}^3$, for some short period of time, starting at $t = 0$. Essentially, what we are trying to establish is the change in the quaternion $q_{\mathrm{m}} q_{\mathrm{s}}^{-1}$ if an infinitesimal rotation $W dt$ is superimposed on top of the original simulation object rotation q_{s}, so that we can compute a formula for how the virtual coupling torque will change. We first use the following formula, valid for for an arbitrary rotating object, and derived, for example, in [BW03]:

$$\frac{dq_{\mathrm{s}}}{dt}_{|t=0} = \frac{1}{2}\hat{W} q_{\mathrm{s}}, \tag{4.19}$$

where $\hat{W}$ denotes a quaternion whose scalar part is zero and whose vector part equals $W \in \mathbb{R}^3$. Note that the multiplication between $\hat{W}$ and q_{s} in the above formula is quaternion multiplication. Then, we have

$$\frac{d}{dt} T_{\mathrm{VC}} = -k_{VC,\mathrm{torque}} \,\mathrm{vector}\Big(\dot{q_s} q_m^{-1}\Big) = \mathrm{vector}\Big(\frac{1}{2}\hat{W}(-k_{VC,\mathrm{torque}} q_{\mathrm{s}} q_{\mathrm{m}}^{-1})\Big) = \frac{1}{2}\mathrm{vector}(\hat{W} q_T), \tag{4.20}$$

where we have denoted $q_T = -k_{VC,\mathrm{torque}} q_{\mathrm{s}} q_{\mathrm{m}}^{-1}$, and where all time derivatives are meant to be taken at $t = 0$. Note that we previously assumed $\mathrm{scalar}(q_T) > 0$, where $\mathrm{scalar}(\_)$ denotes the scalar part of the quaternion, and that we have $T_{VC} = \mathrm{vector}(q_T)$ by Equation 4.14. We can now use the quaternion equality

$$\mathrm{vector}(q_1 q_2) = \mathrm{scalar}(q_2)\mathrm{vector}(q_1) + \mathrm{scalar}(q_1)\mathrm{vector}(q_2) + \mathrm{vector}(q_1) \times \mathrm{vector}(q_2), \tag{4.21}$$

which is valid for any quaternions q_1, q_2, to continue the derivation from Equation 4.20

$$\frac{d}{dt} T_{\mathrm{VC}} = \frac{1}{2}\Big(\mathrm{scalar}(q_T) W + W \times \mathrm{vector}(q_T)\Big) = \frac{1}{2}\Big(\mathrm{scalar}(q_T) I - \widetilde{T_{\mathrm{VC}}}\Big) W. \tag{4.22}$$

On the other hand, we have

$$\frac{d}{dt} T_{\mathrm{VC}} = \frac{\partial T_{\mathrm{VC}}}{\partial \omega}\frac{\partial \omega}{\partial t} = \frac{\partial T_{\mathrm{VC}}}{\partial \omega} W. \tag{4.23}$$

Since Equations 4.23 and 4.22 hold for any $W \in \mathbb{R}^3$, we can now state the final result:

$$\frac{\partial T_{\mathrm{VC}}}{\partial \omega} = \frac{1}{2}\Big(\mathrm{scalar}(q_T) I - \widetilde{T_{\mathrm{VC}}}\Big), \tag{4.24}$$

where we previously defined

$$q_T = -k_{VC,\mathrm{torque}} \,\mathrm{toPositiveHemisphere}\Big(q_{\mathrm{s}} q_{\mathrm{m}}^{-1}\Big), \tag{4.25}$$

$$T_{\mathrm{VC}} = \mathrm{vector}(q_T). \tag{4.26}$$

Large rotations: Equation 4.24 gives the gradient of the virtual coupling torque under *arbitrary* orientations of the manipulandum and the simulation object (except assuming that not in the singular 180 degree rotation configuration). In particular, this formula is not limited to small orientation differences between the manipulandum and the simulation object. If only small orientation differences are considered, the $\widetilde{T_{\mathrm{VC}}}$ term in Equation 4.24 can be omitted, yielding the approximation described in the static virtual coupling VPS paper of [WM03]. Such an approximation gives a virtual coupling torque gradient where an infinitesimal rotation is added on top of another previous infinitesimal rotational difference between the manipulandum and simulation object. Our formulation from Equation 4.24 supports an infinitesimal incremental rotation on top of an existing finite rotational difference. In our simulations, the orientation difference between the manipulandum and the simulation object can be large (often, e.g., 20 degrees or more). In such cases, the extra term $\widetilde{T_{\mathrm{VC}}}$ better captures torque changes if the simulation object is rotated along axes orthogonal to T_{VC}, where the $\widetilde{T_{\mathrm{VC}}}\Delta\omega$ term does not vanish.

4.5 Separating the simulation from haptics

We designed our simulation to run independently from haptics, in the following sense. The simulation happens in its own "simulation world" and could run without an actual haptic device present. The device is only a medium to obtain the manipulandum trajectory and display the computed forces and torques. The simulation receives the manipulandum trajectory as input, one datapoint at a time. This input typically comes from the haptic device, but it can be, for example, read from a file, or obtained via some other input device. The units of length as reported by the haptic device need not be the same as the units used in the simulation; in our simulations, they are almost always different. We use a scaling factor α (adjustable by the user) that controls how many millimeters of the physical haptic device space correspond to one distance unit of the simulation world:

$$x_{\text{in physical device units}} = \alpha \cdot x_{\text{in simulation units}} \ . \tag{4.27}$$

The user can adjust parameter α during the simulation. Suppose the user is exploring a particular scene under varying values of α. Without compromising anything essential, we can simplify the discussion by assuming that virtual coupling is not used, i.e., all the contact forces and torques are rendered directly to the device. If α is large, then for a fixed physical manipulandum displacement, the haptic object in the simulation moves a small distance. Large physical device workspace will be necessary to cover the entire scene if the scene is large. On the upside, for a fixed physical rendered force stiffness (in physical device units, e.g. $0.6N/mm$), the penetration depths incurred by the time the contact force reaches some fixed value (e.g. $8.5N$ on our device) will be small. This is because (1) it always takes $14.17mm$ for the device to ramp up forces from $0N$ to $8.5N$ if stiffness is $0.6N/mm$, and (2) while device travels the $14.17mm$, the object will move little in the simulation world, resulting in a shallow contact penetration. Conversely, small values of α make for a fast traveling manipulandum position in the simulation, so the scene can be easily explored, but the penetration depths incurred during contact will be larger. Middle ground needs to be sought,

such that good scene coverage is achieved, while at the same time penetrations are not too deep. A good choice for α is a value such that the user can about cover the scene, or an interesting subpart of the scene, within the physical workspace of the device. Larger workspaces can be simulated using workspace indexing [CK05b]. We use a simple workspace indexing scheme that works as follows: when the user presses and holds the button on the PHANToM end-effector, they can freely reposition and reorient manipulandum without this having any effect on the manipulandum position and orientation in the simulation world. For example, if the user reaches the boundary of the device workspace (translational or rotational), they can use the button to physically move and orient the physical manipulandum back to the neutral rest manipulandum configuration. Our software internally keeps track of appropriate transformation matrices, so that when the button is released, the manipulandum position and orientation in the simulation world simply continue from where they were before the button was pressed.

All the manipulandum positions and orientations, and all the contact forces and torques in this thesis (other than here in this discussion) refer to the simulation world. The manipulandum position of the haptic device is transformed (virtual indexing + α-scaling) to the corresponding manipulandum position in the simulation world. We then perform deformable object simulation, collision detection and compute contact forces and torques in the world coordinate frame of the simulation world (not to be confused with the local frame of reference of the simulation object). Another scaling factor could be used to convert the simulation forces and torques to the forces and torques that are actually sent to the device; we, however, use a scaling factor of 1 (however, see the virtual coupling saturation discussion in Section 4.9). That is, we interpret the computed forces and torques as if they were given in N and mNm (milliNewton times meter). We do so since this extra scaling factor would not add anything substantially new to our simulations: if one wishes to scale all the rendered forces and torques by a constant β, one can simply scale $k_C, k_{\mathrm{VC}}, k_{\mathrm{VC,torque}}$ by β. Other than scaling the forces and torques rendered to the user, this will have no effect on the simulation, as the factor β simply cancels out when solving for the zero net equilibrium of forces and torques in Equations 4.1 and 4.2.

4.6 Limiting maximum speed of the haptic object

Without restricting velocities, it is possible for the user to propel the object very fast into contact, causing deep penetrations upon impact, which can in turn cause instabilities in the haptic signal. Likewise, without restricting angular velocities, the user can rotate the object very quickly (especially around the roll axis on our PHANToM Premium device), entering fast contact with one part (arm) of the object.

These velocities can be limited as follows (similar ideas were used by others before [MPT06]). The user specifies the maximum velocity and angular velocity that can occur in the simulation. This gives a maximum distance $d_{\max}$ and maximum angle $\theta_{\max}$ that the simulation object can travel and rotate in one haptic cycle. Each time the new position and orientation of the simulation object is determined by solving Equations 4.1 and 4.2 (i.e., once per haptic cycle), we check whether the

new position $x_{\text{grasp,s,new}}$ is more than $d_{\max}$ away from the previous position $x_{\text{grasp,s,old}}$. If yes, the new position $x_{\text{grasp,s}}$ is not set to $x_{\text{grasp,s,new}}$, but instead to the location a distance of $d_{\max}$ away from $x_{\text{grasp,s,old}}$ to $x_{\text{grasp,s,new}}$. In the same way, we impose a limit on maximum rotations during every haptic cycle. If the user moves his/her hand very quickly, causing the manipulandum to exceed maximum velocities, the simulation object will start lagging behind the manipulandum, causing a viscous force/torque to be rendered to the user. This has a convenient side-effect of slowing down the user's hand, thus lowering the velocities. Even if force/torque feedback is turned off and manipulandum is moved around at extremely high speed, the scheme will limit simulation object's displacements within one haptic cycle to $d_{\max}$, which helps with the convergence of static virtual coupling and therefore improves haptic stability.

Also note that the temporal coherence module uses limits $d_{\max}$ and $\theta_{\max}$ to predict the next possible cycle when a currently non-contacting point can enter contact. The lower these limits, the more aggressive the prediction of the temporal coherence module, hence larger temporal coherence benefits. Of course, setting low limits means that the user cannot move the object very fast in the scene, so a tradeoff needs to be sought. In our simulations, we typically set maximum linear velocity to $0.6m/s$ and maximum angular velocity to $360\text{degrees}/s$. With faster computer speeds, one can run the simulation at higher update rates, e.g., several kilohertz, and then higher velocity limits become possible. For example, one can keep $d_{\max}$ constant (which, roughly speaking, implies a fixed level of haptic stability) if both the haptic update rate and the maximum linear velocity are increased five-fold (or by any other factor).

4.7 Controlling maximum stiffness

Haptic devices can only render limited stiffnesses. For example, on an impedance device (that is, a device that senses position and renders force), the rendered force can only increase by a certain device-dependent maximum amount per unit displacement of the manipulandum, or else the device experiences instabilities. On our PHANToM Premium 1.5 6-DOF device, this limit is in the range of $0.6N/mm$. The higher the stiffness, the more challenging the design of the haptic device. Ideally, one would wish to be able to render (close to) infinite stiffnesses, such as pushing against a rigid physical wall. With existing devices, however, it is necessary for the haptic simulation to ensure that the forces sent to the device do not exceed the maximum stiffness, otherwise device instabilities can occur. In this section, we analyze how static virtual coupling limits the maximum stiffness of the forces sent to the device. We limit the discussion to the 3-DOF case: an object manipulated using the three translational degrees of freedom (no rotations) with the simulation computing contact forces (torques are not discussed in this section), which are then sent to the device. We prove that if haptic update rate was unlimited ("continuous time control" [CGSS93]), static virtual coupling guarantees an upper limit on the stiffness of forces that are ever sent to the haptic device.

Multiplane penalty contact: In our penalty-based model, the contact is modeled as a collection of penalty-based springs, with different spatial orientations. At any given moment of time, there are $\ell \geq 0$ points (penalty springs) in contact, with contact normals $n_i, i = 1, \ldots, \ell$. Assuming that the distance field gradient at point i is oriented in the direction of n_i (as described in Section 4.2), then the contact model becomes ℓ penalty contacts against ℓ planes with normals n_i (see Figure 4.4). We call such contact *multiplane penalty contact*. It arises naturally with penalty-based contact between two 3D objects, and it has been used by other researchers as well [Ota04, OL06].

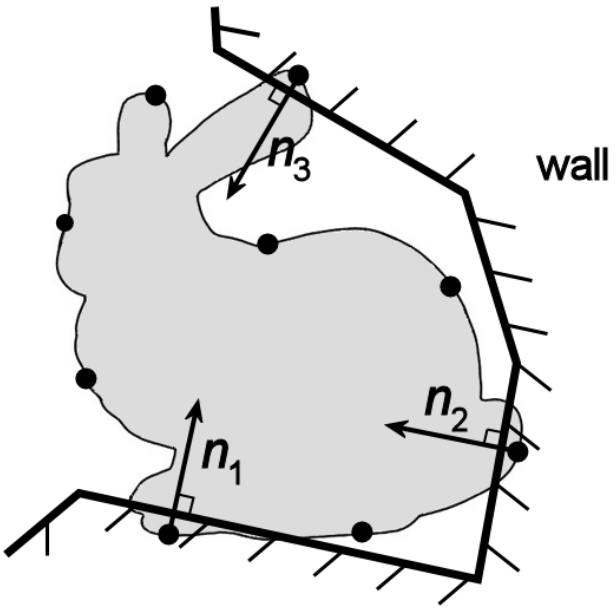


Figure 4.4: **Multiplane penalty contact** ($\ell = 3$).

Let $J = \frac{\partial F_C}{\partial x}$ be the gradient of the contact force (under multiplane penalty contact) with respect to simulation object position. Maximum contact stiffness is defined as the largest increase in force per given simulation object displacement, that is, $\max \|J \cdot dx\| / \|dx\|$, over all directions dx. All norms in this section are 2-norms. Maximum contact stiffness therefore equals $\|J\|_2$, the 2-matrix norm of J. We can now prove the following theorem.

Theorem: Assume a multiplane penalty contact configuration with $\ell \geq 1$ contact points with contact normals n_i, and penalty stiffnesses k_i, for $i = 1, \ldots, \ell$. Then, maximum contact stiffness equals $k_1 + ... + k_\ell$, which is achieved if and only if all normals n_i are equal.
Proof: If $n_i = n$ for all $i = 1, \ldots, \ell$, then we have ℓ parallel springs whose stiffnesses simply add up to $k_1 + ... + k_\ell$. Assume now that contact normals n_i point in arbitrary directions. Then, it follows from the penalty force model that the net change in contact force of spring i due to an object (3D) displacement Δx equals $k_i n_i n_i^T \Delta x$. Therefore, the net contact force change is

$$\Delta F = J \Delta x = -\Big(\sum_{i=1}^{\ell} k_i n_i n_i^T \Big) \Delta x. \tag{4.28}$$

Since n_i is a unit vector, we have $\| - k_i(n_i^T \Delta x) n_i \| = k_i |n_i^T \Delta x| \leq k_i \|\Delta x\|$, and therefore $\|\Delta F\| \leq \left(\sum_{i=1}^{\ell} k_i\right) \|\Delta x\|$, i.e., maximum contact stiffness equals $k_1 + ... + k_\ell$. It remains to be proven that this maximum bound can only be achieved if all n_i are equal. Assume maximum bound is achieved for some normals n_i. Then, there exists a *unit* vector direction Δx, such that

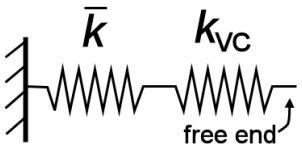
$$\| - \left(\sum_{i=1}^{\ell} k_i n_i n_i^T\right) \Delta x \| = k_1 + \ldots + k_\ell. \tag{4.29}$$

By denoting $a_i := -k_i n_i n_i^T \Delta x$ we can rewrite this as $\| \sum_{i=1}^{\ell} a_i \| = k_1 + \ldots + k_\ell$. At the same time, we also have $\|a_i\| \leq k_i$, so it follows from the triangle inequality that

$$\sum_{i=1}^{\ell} \|a_i\| \geq \| \sum_{i=1}^{\ell} a_i \| = k_1 + \ldots + k_\ell \geq \sum_{i=1}^{\ell} \|a_i\|. \tag{4.30}$$

In other words, we have equality in the triangle inequality. This is only possible if all vectors a_i are positive scalar multiples of one another, which is only possible if $n_i = n$ for all i. QED

In our simulations, we usually set $k_i = k_C$ for all i. Therefore, the contact stiffness will be $\ell\, k_C$ when ℓ points are in parallel contact, so contact stiffness can grow arbitrarily large as more and more points are in contact. For example, if a surface is sampled with progressively finer pointshells, stiffness would grow out of control. Even if points are weighted with their corresponding surface areas, stiffness can become large if the contact area is large. This problem is addressed using virtual coupling. Say that one connects two springs serially, one with an acceptable stiffness k_{VC} (the virtual coupling spring), and another with unpredictable stiffness $\overline{k} = \ell k_C$ (the contact forces). Then, it is an easy exercise to show that the net stiffness felt at the free end is $1/(1/k_{VC} + 1/\overline{k})$. This is always less than or equal to $\min\{k_{VC}, \overline{k}\}$. Therefore, in a 3-DOF simulation, the maximum stiffness felt by the user if ℓ points are in contact is



$$\frac{1}{1/(\ell \cdot k_{\mathrm{C}}) + 1/k_{\mathrm{VC}}}, \tag{4.31}$$

where k_{VC} is the stiffness of the translational virtual coupling. Therefore k_{VC} presents an upper bound on the stiffness ever rendered to the haptic device, regardless of ℓ.

The previous paragraph gave an informal discussion on maximum stiffness. We have also proved the following theorem that formally proves a maximum limit on stiffness under virtual coupling (for a three-dimensional 3-DOF simulation):

Theorem: Assume a 3-DOF simulation with continuous time control, and a multiplane penalty contact configuration with $\ell \geq 1$ contact points with arbitrary contact normals n_i. Let $J = \frac{\partial F_C}{\partial x}$ be the gradient of the contact force with respect to simulation object position. Let $k_{\mathrm{VC}} \in \mathbb{R}$ denote the stiffness of translational static virtual coupling. Then (1) the 3×3 linear system of equations

to be solved at every timestep of static virtual coupling (the 3×3 version of the system from Equations 4.1, 4.2) is always non-singular (even if J is singular), and (2) the maximum stiffness rendered to the device is

$$\frac{1}{\frac{1}{\|J\|_2} + \frac{1}{k_{\mathrm{VC}}}}, \tag{4.32}$$

which is always less than or equal to k_{VC}.

Proof: Throughout the proof, remember that for a symmetric matrix A the maximum absolute eigenvalue equals the matrix norm $\|A\|_2$. Suppose the simulation object is initially in static equilibrium against the virtual coupling force and contact forces, and suppose the user then displaces the manipulandum by a small displacement Δx. The virtual coupling force on the simulation object therefore changes by $k_{\mathrm{VC}}\Delta x$. Since the simulation object was previously in equilibrium, the net force on the simulation object now equals $k_{\mathrm{VC}}\Delta x$. We need to compute the change in the equilibrium position of the virtual object, i.e., we need to compute such a displacement $(\Delta x)_{\mathrm{SO}}$ of the simulation object that the net force on the simulation object vanishes again:

$$k_{\mathrm{VC}}\Delta x + J(\Delta x)_{\mathrm{SO}} - k_{\mathrm{VC}}(\Delta x)_{\mathrm{SO}} = 0. \tag{4.33}$$

The middle term in the above equation corresponds to the change in contact forces on the simulation object, and the last term corresponds to the change in the virtual coupling force on the simulation object. The equation can be rewritten as

$$\Big(J - k_{\mathrm{VC}} I_3\Big)(\Delta x)_{\mathrm{SO}} = -k_{\mathrm{VC}}\Delta x, \tag{4.34}$$

where I_3 is the 3×3 identity matrix. This 3×3 system of equations is precisely the 3×3 version of the system from Equations 4.1, 4.2. Its nonsingularity can be established as follows. From the multiplane penalty contact model it follows that

$$J = -\sum_{i=1}^{\ell} k_i n_i n_i^T, \tag{4.35}$$

and therefore J is symmetric negative-definite. By the previous theorem the spectrum of J is contained inside $[-\|J\|_2, 0] = [-k_1 - \ldots - k_\ell, 0]$. The spectrum of $J - k_{\mathrm{VC}} I_3$ is therefore contained inside $[-\|J\|_2 - k_{\mathrm{VC}}, -k_{\mathrm{VC}}]$. Because $k_{\mathrm{VC}} > 0$, the matrix in the system of equations 4.34 is therefore invertible, which proves (1).

To prove (2), observe that the net change in the rendered force to the user as the manipulandum displaces Δx and the virtual object consequently displaces by $(\Delta x)_{\mathrm{SO}}$ will be

$$k_{\mathrm{VC}}\Big((\Delta x)_{\mathrm{SO}} - \Delta x\Big) = -k_{\mathrm{VC}}\Big(k_{\mathrm{VC}}(J - k_{\mathrm{VC}} I_3)^{-1} + I_3\Big)\Delta x = A\Delta x, \tag{4.36}$$

where we have denoted

$$A = -k_{\mathrm{VC}}\Big(k_{\mathrm{VC}}(J - k_{\mathrm{VC}} I_3)^{-1} + I_3\Big). \tag{4.37}$$

The stiffness rendered to the user equals $\|k_{\mathrm{VC}}\Big((\Delta x)_{\mathrm{SO}} - \Delta x\Big)\|/\|\Delta x\|$. Therefore, the maximum stiffness rendered to the user is bounded by $\|A\|_2$. Matrix A is symmetric and its spectrum can be bounded as follows. As demonstrated previously, the spectrum of $J - k_{\mathrm{VC}}I_3$ is contained inside $[-\|J\|_2 - k_{\mathrm{VC}}, -k_{\mathrm{VC}}]$, therefore the spectrum of $(J - k_{\mathrm{VC}}I_3)^{-1}$ is contained inside $[-1/k_{\mathrm{VC}}, -1/(\|J\|_2 + k_{\mathrm{VC}})]$. It follows that the spectrum of A is contained inside the interval

$$\Big[0, \frac{k_{\mathrm{VC}}\|J\|_2}{k_{\mathrm{VC}} + \|J\|_2}\Big] = \Big[0, \frac{1}{\frac{1}{\|J\|_2} + \frac{1}{k_{\mathrm{VC}}}}\Big]. \tag{4.38}$$

Therefore

$$\|A\|_2 \leq \frac{1}{\frac{1}{\|J\|_2} + \frac{1}{k_{\mathrm{VC}}}}, \tag{4.39}$$

which proves (2). QED

Our theorem establishes that virtual coupling regularizes the contact Jacobian: it makes for a non-singular system of static virtual coupling equations even when the contact Jacobian J is degenerate (which is easily the case; e.g., it is always the case when $\ell \leq 2$). The theorem also proves an upper limit on the rendered stiffness of multiplane penalty contact if collision detection and contact force computation could be performed infinitely fast (continuous time control). In practice, however, discretization introduces additional instabilities. Stability is improved if k_{C} is scaled down when there are multiple points in contact; as in VPS, we replace k_{C} with $L \cdot k_{\mathrm{C}}/\ell$ (we use $L = 10$) if $\ell \geq L$ points are in contact.

4.8 Static damping

There is no dissipation in the standard static virtual coupling model, which can manifest as slight instabilities in some cases, e.g., if one object quickly slides around the other object. We augment the static virtual coupling model by introducing static damping: after $\Delta x, \Delta\omega$ are computed, we only change the simulation position and rotation by $(1-\alpha)\Delta x, (1-\alpha)\Delta\omega$, where $\alpha \in [0, 1)$ controls the amount of static damping. Such damping restores the simulation position exponentially to the manipulandum position, similar to critical damping in mass-spring-damper dynamic virtual coupling models [MPT99]. We set $\alpha = 0.5$ in all our simulations; high values of α are avoided since they introduce a surface stickiness effect.

4.9 Virtual coupling saturation

The displacement-force relationship of virtual coupling is designed to be linear only up to a certain maximum force value, after which the force is held constant (saturates) at some maximum value, even if the user stretches virtual coupling even further. Such saturation prevents the user from pulling the manipulated object into the interior of the other object. Even if force feedback is turned off completely, and the user is free to position the haptic manipulandum arbitrarily, the virtual

object is prevented from penetrating the environment more than allowed by the maximum value of the virtual coupling spring. The value to which the force saturates is typically set such that it corresponds to some shallow penetration, such as, for example, half a voxel. This enables us to control the amount of maximum penetration. Saturation as such acts as a *virtual proxy*, similar to the approaches of [ZS95, RKK97].

Virtual coupling saturation worked very well for us in practice, however, strictly speaking, it does not guarantee theoretically that deep penetrations cannot occur. The reason for this is that manipulandum position could, theoretically speaking, vary large distances from one haptic cycle to the next. For example, imagine a thin wall, and suppose the manipulandum is first located in free space on one side, but then suddenly moves to free space on the other side in one haptic cycle. The simulation object will then, in principle, skip through the wall. However, the problem is alleviated by imposing a limit on simulation object position and orientation change within one haptic cycle, as discussed in Section 4.6. If the wall is thicker than $d_{\max}$ from Section 4.6, pop-through will not occur, no matter how fast the manipulandum moves. Also, if one increases the haptic update rate, say, by a factor of 2, one will have sampled an additional manipulandum position in the middle of the way from one side of the wall to the other side, which will prevent the pop-through. Ultimately, the right way to prevent pop-through is to perform continuous collision detection; however, existing deformable algorithms would not scale simultaneously to both haptic rates and the complexity of our examples.

We implemented saturation for the virtual coupling force, and an extension to torques would be possible too. The idea of saturation is to keep the force linear for small displacements, but then smoothly saturate it to some maximum constant value. We use the saturation method presented in [WM03], with some modifications. We explicitly give the virtual coupling force gradients, and show how the net force/torque equilibrium Equations 4.1 and 4.2 can be solved in the presence of degenerate virtual coupling force gradients in the saturated region (in Section 4.10).

Saturation works as follows. The user specifies the maximum permitted penetration depth, in distance field voxel units. We then compute the contact force felt by a single point under this penetration depth, which is set to be the plateau (maximum value) of the saturated virtual coupling force; call it $F_{\mathrm{VC,max}}$. Note that if more than one point is in contact, the situation in terms of penetration becomes better, because the penalty forces of several simultaneous contact points only admit progressively shallower penetrations. The relationship between virtual coupling spring displacement and virtual coupling force is set to be linear up to $F_{\mathrm{VC,max}}/2$, i.e., the maximum displacement up until which the virtual coupling force is linear is chosen to be $r_{\mathrm{lin}} = F_{\mathrm{VC,max}}/(2k_{\mathrm{VC}})$. Beyond r_{lin}, our virtual coupling force saturates exponentially to the limit $F_{\mathrm{VC,max}}$, at a rate such that the derivative of the force is continuous at the joint at r_{lin} :

$$F_{\mathrm{VC}} = -f(r)e_r, \quad \text{where} \quad f(r) = \begin{cases} k_{VC}\, r\ ; & \text{if} \quad r \le r_{\mathrm{lin}} \\ F_{\mathrm{VC,max}}\Big(1 - \frac{1}{2}\Big(2k_{\mathrm{VC}}(r_{\mathrm{lin}} - r)/F_{\mathrm{VC,max}}\Big)\Big)\ ; & \text{if} \quad r > r_{\mathrm{lin}}. \end{cases} \tag{4.40}$$

Here, F_{VC} is the virtual coupling force on the simulation object, e_r is a unit vector in the direction of $x_{\mathrm{grasp,s}} - x_{\mathrm{grasp,m}}$, and $r = \|x_{\mathrm{grasp,s}} - x_{\mathrm{grasp,m}}\|$. An alternative choice for saturation would be to

use a cubic spline [OL05], which (say) reaches the maximum value of $F_{\mathrm{VC,max}}$ at $r = 3r_{\mathrm{lin}}$, with a continuous derivative at both joints $r = r_{\mathrm{lin}}, r = 3r_{\mathrm{lin}}$.

Gradient of the saturated virtual force: In the linear region, the computation can proceed just as outlined in Section 4.4. For $r > r_{\mathrm{lin}}$, some algebra using spherical coordinates gives

$$\frac{\partial F_{\mathrm{VC}}}{\partial x} = -\frac{f(r)}{r} I + \Big(\frac{f(r)}{r} - f'(r)\Big) e_r \otimes e_r, \tag{4.41}$$

where $a \otimes b = ab^T$ for $a, b \in \mathbb{R}^3$. This expression can be directly evaluated at runtime. Note that as $r \to \infty$, the gradient becomes progressively singular, as $\frac{\partial F_{\mathrm{VC}}}{\partial x} e_r$ tends to zero much faster than the gradient in the other two orthogonal directions. Consequently, the 6×6 linear system of Equations 4.1 and 4.2 can become singular. We deal with this issue in Section 4.10.

Rendering the saturated force to the device: If saturation is not used, then the force that is rendered to the device is $-F_{\mathrm{VC}}$. If using saturation, one can of course still render $-F_{\mathrm{VC}}$, which means that the actual physical force felt by the user will also saturate, together with the simulation virtual coupling force. However, in practice, this maximum rendered force often turns out to be significantly less than the maximum force that a particular haptic device can render, and then large forces are never rendered. We address this by (1) computing F_{VC} according to Equation 4.40, and use this value everywhere in the simulation, and (2) always rendering $-k_{\mathrm{VC}} \cdot r$ to the device, even if $r > r_{\mathrm{lin}}$. If the force exceeds the maximum force renderable by the particular device ($8.5N$ for our device), then we clamp the rendered force at this maximum value. Under this scheme, the relationship between the simulation virtual coupling force and the force rendered to the device is no longer linear. We did not detect any problems in practice. The relationship is monotonic, and linear for small forces.

The rendered force is only C^0 continuous when it hits the maximum plateau ($8.5N$). On our PHANToM device, we are unable to feel this C^1 discontinuity. If necessary, the discontinuity could be avoided by applying another nonlinear filter on $-k_{\mathrm{VC}} \cdot r$ prior to rendering. This filter would be identity for small forces, followed by a smooth ramp to the maximum renderable force value. This could be done using an exponential function or cubic spline, in the same spirit as we saturated F_{VC}.

Saturated torques: The virtual coupling torques can be saturated in a similar way as the virtual coupling forces. We did not implement this extension due to additional implementation complexity. If virtual coupling torque is not saturated, it is possible to apply large virtual coupling torques, which can in some cases lead to penetrations deeper than what allowed by saturated virtual coupling forces, or even pop-through. For example, in our implementation of the path planning alpha puzzle, there is a configuration (see Figure 4.5, left) where the user can apply large virtual coupling torques to "cheat" the puzzle. While we did not find this spurious shortcut to be particularly distracting, it could be remedied by saturating virtual coupling torques. Note, however, that saturating torques alone is not always sufficient due to mechanical leverage considerations. For example,

imagine that the user is manipulating a long rod, with the point of attachment at the rod's center of mass (see Figure 4.5, right). Suppose that the rod is positioned horizontally, and is in contact with a narrow vertical pillar at one of its ends. For a fixed applied virtual coupling torque, in order to prevent pop-through, the pillar has to resist with a contact force that grows linearly in the length of the rod. For a fixed virtual coupling torque saturation limit, there is always a rod long enough that a pop-through could occur. This problem can be addressed by choosing a torque saturation limit that takes into account the haptic object's diameter, in addition to the maximum allowable distance field penetration depth.

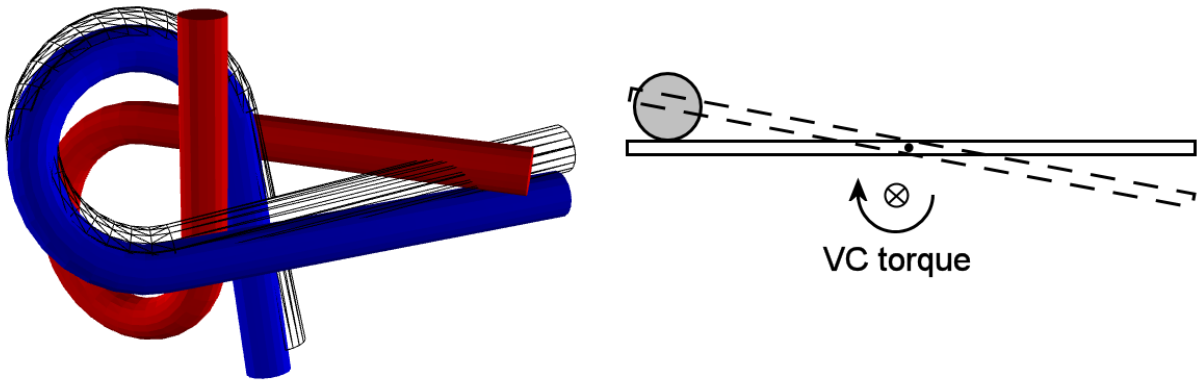


Figure 4.5: **Virtual coupling torque saturation:** Left: without torque saturation, the user can squeeze the blue alpha in between the other (red) alpha's gap, by using the blue alpha's long end as a lever. Right: A fixed torque saturation limit will overpower contact penalty forces for long objects, such as for this manipulated long (horizontally positioned) rod in contact with a static vertical cylinder (view from top).

4.10 How to handle degenerate virtual coupling gradients

In certain degenerate contact configurations, the gradients of the contact force with respect to haptic object position, or orientation, become rank deficient. For example, when in free space, both gradients are identically zero. If a square box is positioned in an (infinite) shaft with a square profile, there is no contact force resistance to the motion in the direction along the shaft, hence the gradient of the contact force with respect to simulation object position will have a null space of dimension 1. Similarly, there will be a degeneracy of dimension 2 if a square box is positioned in between two infinite planes, in simultaneous contact with both planes. The contact torque gradients can be degenerate too: imagine a ball contacting a hemispherical hole obtained by carving the shape of the ball from a wall. In this case, contact torques offer no resistance to rotation at all, i.e., this is a contact torque degeneracy of dimension 3. A torque degeneracy of dimension 1 is possible too: consider a cylinder in a cylindrically shaped shaft. If one positions a sphere in such a

shaft, one obtains a degeneracy of dimension 4. In summary, degenerate contact force and torque gradients are possible even if not in free space.

In practice, any such degeneracy is removed when the virtual coupling gradients are added to the 6×6 system of Equations 4.1 and 4.2. In this sense, virtual coupling regularizes the system of equations to be solved, making it nonsingular even in the presence of degenerate contact configurations. However, special care has to be taken if using virtual coupling saturation, because in this case the gradient of the virtual coupling force itself vanishes in the saturated region, and therefore cannot regularize any degenerate contact gradients. Essentially, the net force/torque gradient when moving in singular directions is zero, and as such the simulation, observing only the current net forces and torques and the current gradients, cannot know where to move the object to minimize the net force/torque imbalance. If no special care is taken, this leads to an (almost) singular system of Equations 4.1 and 4.2, potentially leading to simulation "explosion". For example, this could happen in free space if the user moves the haptic handle so quickly that the displacement between the manipulandum and the haptic object grows to the fully saturated region. Imposing limits on velocities typically does not help, as in practice (in our examples) those limits are too large to remedy this problem. The problem is not limited only to free space, however. For example, for the square box in a square infinite shaft, one can reproduce the same problem if moving the manipulandum very fast along the axis of the shaft.

We designed a scheme that handles these kind of degeneracies, and guarantees stable operation. This scheme is only necessary if virtual coupling saturation is used (and typically it is used in our demos), and it works as follows. Let A be the 6×6 system matrix from Equations 4.1 and 4.2. Call the right-hand side $b \in \mathbb{R}^6$, and the unknown vector $x = (\Delta x, \Delta\omega) \in \mathbb{R}^6$. Instead of proceeding to solve $Ax = b$ directly via Gaussian elimination (which could fail as we do not know whether A is singular or not), we first compute the singular value decomposition of $A = U\Sigma V^T$, where $U, V, \Sigma \in \mathbb{R}^{6\times6}$, U and V are 6×6 orthogonal matrices, and Σ is a diagonal matrix of non-negative singular values $\sigma_1 \geq \sigma_2 \ldots \sigma_6 \geq 0$. We threshold the singular values: any singular value smaller than $\epsilon\sigma_1$ is considered too close to zero. We use $\epsilon = 0.001$ everywhere in our implementations. Let V_{nd} denote the submatrix of V corresponding to the retained singular values. The columns of V_{nd} span our non-degenerate solution space. Likewise, the remaining columns of the matrix V (denote the submatrix by V_{d}) span our degenerate solution space, yielding a decomposition $V = [V_{\mathrm{nd}} \,|\, V_{\mathrm{d}}]$. The matrix U is decomposed in the same way.

We decompose the problem into the non-degenerate and degenerate part. For the non-degenerate subspace, there is sufficient (i.e., nonsingular) information on gradients of the net equilibrium forces and torques. Therefore, we compute the minimum norm solution of $Ax = b$ within the subspace spanned by V_{nd}, i.e., $x_{\mathrm{nd}} = V_{\mathrm{nd}}\Sigma^{-1}U_{\mathrm{nd}}^T b$. For the degenerate subspace, the system is offering negligible resistance in the directions of the subspace. Therefore, we simply pick the x_{d} from the degenerate subspace which minimizes the difference between the manipulandum position and simulation object position, i.e., we set

$$x_{\mathrm{d}} = V_{\mathrm{d}}V_{\mathrm{d}}^T\Big(x_{\mathrm{grasp,m}} - x_{\mathrm{grasp,s}}\Big). \tag{4.42}$$

We use $x = x_{\mathrm{nd}} + x_{\mathrm{d}}$ as the final solution. Note that this scheme is different from Truncated

Singular Value Decomposition (TSVD) and also from Tikhonov regularization [TV77]. TSVD would use $x_d = 0$. In Tikhonov regularization, one would change the truncated singular values to some constant ϵ', and then solve the regularized system of equations. It is not obvious how to select a stable ϵ', as too small values cause solution overshoot and instabilities, whereas too large values cause the simulation object to lag behind the manipulandum, at smaller velocities than with typical velocity limits of Section 4.6.

Let us give an example of how our scheme works. Suppose the user moves the manipulandum very quickly in free space. Then, V_{nd} will span the 3-dimensional space of infinitesimal rotations, whereas V_{d} will span the degenerate 3-dimensional space of translations. The simulation object will snap to the manipulandum within the subspace spanned by V_{d}. In the case of a square box in a square shaft, if the user moves the box very quickly along the axis of the shaft, V_{d} will span the direction along the shaft. Within V_{d}, the simulation object will snap to the manipulandum. In the other five directions, the simulation behaves as usual. If the user moves the box in the plane perpendicular to the shaft axis, or attempts to rotate the box, they will feel the contact forces and torques, while simultaneously feeling no resistance to the motion along the shaft axis.

Note that x_{d} will be invoked only when the rank is degenerate. In the vast majority of cases, there will be no singularity and only the x_{nd} term will be present. This algorithm requires performing a SVD on a 6×6 matrix, as opposed to simply solving the system with Gaussian elimination. SVD is slower than Gaussian elimination, but this was not a significant bottleneck in our simulations: solving the 6×6 linear system with Gaussian elimination typically took 5 microseconds on the machine of Table 3.1, whereas the entire degenerate rank support algorithm, including SVD, typically took 30 microseconds, which is still only 3% of the entire computation time available.

The velocity-limiting scheme of Section 4.6 can still be used, even when using degenerate rank handling. We do this by first computing $x = x_{\mathrm{nd}} + x_{\mathrm{d}}$, then perform the velocity (and angular velocity) limiting on x as usual. Typically our velocity limits are larger than velocities than cause degenerate ranks, so x computed in a degenerate situation will often pass the velocity limiting stage unmodified.

Conclusion

This thesis proposed a class of reduced deformable objects that can undergo large deformations and can be timestepped at high update rates. The reduced dynamics are simulated in a low-dimensional basis of deformations, obtained by appropriately reducing large-deformation (geometrically non-linear) FEM deformable models. We use reduced models because general FEM models do not run at haptics for models with complex geometry. We demonstrated that it is now possible to simulate distributed contact between reduced-deformable objects at haptic rates, with an algorithm that scales to complex geometry. We introduced the nested point-tree, a novel datastructure that can be used to simulate time-critical (deformable) contact. We gave a detailed description of our 6-DOF haptic rendering pipeline, exposing several important details that were previously either sparsely documented or not documented in literature, but are essential for building a robust haptic rendering system. We demonstrated that the entire simulation pipeline can benefit from reduction, not just deformable object simulation. Collision detection and contact largely benefit from our output-sensitive nested point-tree traversal. A general FEM deformable model would explicitly compute the deformations of all vertices (points) at every timestep, whereas we can timestep the deformations in a subspace and therefore do not have to explicitly construct the deformations of all vertices (points) at every haptic cycle. We only have to evaluate individual deformed point position, normal and bounding sphere radius if this particular point is encountered by the tree traversal. Such selective individual evaluation is only possible due to reduction. The reduced deformable model, (reduced-deformable) distance fields and the nested bounded deformation point-tree can be integrated into one system which can render complex geometry in reduced-deformable distributed contact at haptic rates for the first time.

Acknowledgements

This research was sponsored by the National Science Foundation (CAREER-0430528), the Boeing Company, the Link Foundation fellowship to Jernej Barbič, and by generous donations from Pixar, Nvidia, Intel, and Maya licenses donated by Autodesk. We also thank Guido Dhondt for help with his solid mechanics package CalculiX.

Bibliography

[ABA] ABAQUS. ABAQUS Inc., http://www.hks.com.

[AH98a] Richard J. Adams and Blake Hannaford. A Two-Port Framework for the Design of Unconditionally Stable Haptic Interfaces. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1254–1259. IEEE, 1998.

[AH98b] Oliver Astley and Vincent Hayward. Multirate Haptic Simulation Achieved by Coupling Finite Element Meshes Through Norton Equivalents. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1998.

[Alg] Algorithms and Application Group (Nancy Amato), Parasol Lab, Texas A&M University. Motion path planning puzzles (benchmarks) online repository, http://parasol.tamu.edu/groups/amatogroup/benchmarks/mp.

[ASB78] B. O. Almroth, P. Stern, and F. A. Brogan. Automatic Choice of Global Shape Functions in Structural Analysis. *AIAA Journal*, 16(5):525–528, 1978.

[BA05] J. Bærentzen and H. Aanæs. Signed distance computation using the angle weighted pseudo-normal. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):243–253, 2005.

[Bas01] C. Basdogan. Real-time Simulation of Dynamically Deformable Finite Element Models Using Modal Analysis and Spectral Lanczos Decomposition Methods. In *Medicine Meets Virtual Reality (MMVR'2001)*, pages 46–52, 2001.

[BBM+01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 425–432, August 2001.

[Bel01] Ted Belytschko. *Nonlinear Finite Elements for Continua and Structures*. Wiley, 2001.

[BFA02] Robert Bridson, Ronald P. Fedkiw, and John Anderson. Robust Treatment of Collisions, Contact, and Friction for Cloth Animation. *ACM Trans. on Graphics*, 21(3):594–603, 2002.

[BFSV06] Peter Benner, Roland W. Freund, Danny C. Sorensen, and Andras Varga. Special Issue on Order Reduction of Large-Scale Systems. *Linear Algebra and its Applications*, 415:231–234, june 2006.

[BJ05] Jernej Barbič and Doug L. James. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3), August 2005.

[BNC96] Morten Bro-Nielsen and Stephane Cotin. Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. *Comp. Graphics Forum*, 15(3):57–66, 1996.

[BSA00] O. Burchan Bayazit, Guang Song, and Nancy M. Amato. Enhancing Randomized Motion Planners: Exploring with Haptic Hints. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 529–536, 2000.

[BW92] David Baraff and Andrew Witkin. Dynamic Simulation of Non-penetrating Flexible Bodies. *Computer Graphics (Proc. of ACM SIGGRAPH 92)*, 26(2):303–308, 1992.

[BW97] Javier Bonet and Richard D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, 1997.

[BW98] David Baraff and Andrew P. Witkin. Large Steps in Cloth Simulation. In *Proc. of ACM SIGGRAPH 98*, pages 43–54, July 1998.

[BW03] David Baraff and Andrew Witkin. Physically based modelling course notes, 2003.

[CDA99] S. Cotin, H. Delingette, and N. Ayache. Realtime Elastic Deformations of Soft Tissues for Surgery Simulation. *IEEE Trans. on Vis. and Comp. Graphics*, 5(1):62–73, 1999.

[CG98] Marie-Paule Cani-Gascuel. Layered Deformable Models with Implicit Surfaces. In *Graphics Interface*, pages 201–208, 1998.

[CGC+02a] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. A Multiresolution Framework for Dynamic Deformations. In *Proc. of the Symp. on Comp. Animation 2002*, pages 41–48, 2002.

[CGC+02b] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive Skeleton-Driven Dynamic Deformations. *ACM Trans. on Graphics*, 21(3):586–593, July 2002.

[CGSS93] J.E. Colgate, P.E. Grafing, M.C. Stanley, and G. Schenkel. Implementation of Stiff Virtual Walls in Force-Reflecting Interfaces. In *Proc. of IEEE Virtual Reality Annual Int. Symp.*, pages 202–208. IEEE, 1993.

[CK05a] Min Gyu Choi and Hyeong-Seok Ko. Modal Warping: Real-Time Simulation of Large Rotational Deformation and Manipulation. *IEEE Trans. on Vis. and Comp. Graphics*, 11(1):91–101, 2005.

[CK05b] Francois Conti and Oussama Khatib. Spanning large workspaces using small haptic devices. In *Proc. of the World Haptics Conference*, pages 183–188, 2005.

[CM69] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th National Conference of the ACM*, pages 157–172, 1969.

[CMJ95] J.E. Colgate, M.C.Stanley, and J.M.Brown. Issues in the Haptic Display of Tool Use. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 140–145. IEEE, 1995.

[DA04] C. Duriez and C. Andriot. A multi-threaded approach for deformable/rigid contacts with haptic feedback. In *Proc. of the Symp. on Haptic Interfaces For Virtual Environment and Teleoperator Systems*, pages 272–279, 2004.

[DDCB01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling. In *Proc. of ACM SIGGRAPH 2001*, pages 31–36, August 2001.

[DDKA06] Christian Duriez, Frédéric Dubois, Abderrahmane Kheddar, and Claude Andriot. Realistic Haptic Rendering of Interacting Deformable Objects in Virtual Environments. *IEEE Trans. on Vis. and Comp. Graphics*, 12(1):36–47, 2006.

[DSB99] Mathieu Desbrun, Peter Schröder, and Al Barr. Interactive Animation of Structured Deformable Objects. In *Graphics Interface '99*, pages 1–8, June 1999.

[DSP06] Kevin G. Der, Robert W. Sumner, and Jovan Popović. Inverse kinematics for reduced deformable models. *ACM Transactions on Graphics*, 25(3):1174–1179, 2006.

[DW03] G. Dhondt and K. Wittig. CalculiX: A Free Software Three-Dimensional Structural Finite Element Program, 2003.

[ELPZ97] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Trans. on Image Processing*, 6(9):1305–1315, 1997.

[Eri01] Eric de Castro Lopo. Faster Floating Point to Integer Conversions, http://mega-nerd.com/FPcast/, 2001.

[Fis92] J. Fish. The s-version of the finite element method. *Computers and Structures*, 43(3):539–547, 1992.

[Fun77] Y. Fung. *A First Course in Continuum Mechanics*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[FvdPT97] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic Free-Form Deformations for Animation Synthesis. *IEEE Trans. on Vis. and Comp. Graphics*, 3(3):201–214, 1997.

[GKJ+05] Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Transactions on Graphics*, 24(3):991–999, August 2005.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Trans. on Graphics*, 21(3):281–290, July 2002.

[GL89] Alan George and Joseph W. H. Liu. The Evolution of the Minimum Degree Ordering Algorithm. *SIAM Review*, 31(1):1–19, 1989.

[GM97] S. Gibson and B. Mirtich. A Survey of Deformable Modeling in Computer Graphics. Technical Report TR-97-19, Mitsubishi Electric Research Lab, Cambridge, MA, November 1997.

[GME+00] Arthur Gregory, Ajith Mascarenhas, Stephen Ehmann, Ming Lin, and Dinesh Manocha. Six degree-of-freedom haptic display of polygonal models. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 139–146, 2000.

[GTH98] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models. In *Proc. of ACM SIGGRAPH 98*, pages 9–20, July 1998.

[GW06] Joachim Georgii and Ruüdiger Westermann. A multigrid framework for real-time simulation of deformable bodies. *Computer & Graphics*, 30:408–415, 2006.

[HFL00] G. Hirota, S. Fisher, and M. C. Lin. Simulation of Non-penetrating Elastic Bodies Using Distance Fields. Technical report, University of North Carolina at Chapel Hill, NC, April 2000.

[HFS+01] Gentaro Hirota, Susan Fisher, Andrei State, Chris Lee, and Henry Fuchs. An implicit finite element method for elastic solids in contact. In *Proc. of Computer Animation, Seoul, South Korea, 2001*, 2001.

[Hol00a] J. M. Hollerbach. Some current issues in haptics research. In *Proc. IEEE Intl. Conf. Robotics and Automation*, pages 757–762, April 24-28 2000.

[Hol00b] Gerhard A. Holzapfel. *Nonlinear Solid Mechanics*. Wiley, 2000.

[Hop94] Hugues Hoppe. *Surface reconstruction from unorganized points*. PhD thesis, Department of Comp. Science and Engineering, University of Washington, 1994.

[HSO03] K. K. Hauser, C. Shen, and J. F. O'Brien. Interactive Deformation Using Modal Analysis with Constraints. In *Proc. of Graphics Interface*, 2003.

[Hub95] Philip M. Hubbard. *Collision Detection for Interactive Graphics Applications*. PhD thesis, Department of Comp. Science, Brown University, 1995.

[HW04a] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, Berlin, third edition, 2004.

[HW04b] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, Berlin, third edition, 2004.

[IC85a] Sergio R. Idelsohn and Alberto Cardona. A Load-dependent Basis for Reduced Nonlinear Structural Dynamics. *Computers and Structures*, 20(1-3):203–210, 1985.

[IC85b] Sergio R. Idelsohn and Alberto Cardona. A Reduction Method for Nonlinear Structural Dynamic Analysis. *Computer Methods in Applied Mechanics and Engineering*, 49:253–279, 1985.

[ITF04] G. Irving, J. Teran, and R. Fedkiw. Invertible Finite Elements for Robust Simulation of Large Deformation. In *Proc. of the Symp. on Comp. Animation 2004*, pages 131–140, 2004.

[Jam01] D.L. James. *Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects and Other Physical Systems in Equilibrium*. PhD thesis, Institute of Applied Mathematics, University of British Columbia, Vancouver, British Columbia, Canada, 2001.

[JBS06] M. Jones, J. Bærentzen, and M. Sramek. 3d distance fields: a survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[JBT04] Doug L. James, Jernej Barbič, and Christopher D. Twigg. Squashing Cubes: Automating Deformable Model Construction for Graphics. In *Proc. of ACM SIGGRAPH Sketches and Applications*, August 2004.

[JCC06] S. Jun, J. Choi, and M. Cho. Physics-based s-adaptive haptic simulation for deformable object. In *Proc. of the Symp. on Haptic Interfaces For Virtual Environment and Teleoperator Systems*, pages 72–78, 2006.

[JF03] Doug James and Kayvon Fatahalian. Precomputing Interactive Dynamic Deformable Scenes. In *Proc. of ACM SIGGRAPH 2003*, pages 879–887. ACM, 2003.

[JJ] J. Barbič and D. James. The project webpage for the Barbič-James 2005 SIGGRAPH paper, http://graphics.cs.cmu.edu/projects/stvk.

[Jol86] I. T. Jolliffe. *Principal Component Analysis*. New York: Springer-Verlag, 1986.

[JP99] Doug L. James and Dinesh K. Pai. ARTDEFO: Accurate Real Time Deformable Objects. In *Proc. of ACM SIGGRAPH 99*, volume 33, pages 65–72, 1999.

[JP01] Doug L. James and Dinesh K. Pai. A Unified Treatment of Elastostatic Contact Simulation for Real Time Haptics. *Haptics-e, The Electronic J. of Haptics Research (www.haptics-e.org)*, 2(1), September 2001.

[JP02] Doug L. James and Dinesh K. Pai. DyRT: Dynamic Response Textures for Real Time Deformation Simulation With Graphics Hardware. *ACM Trans. on Graphics*, 21(3):582–585, July 2002.

[JP03] Doug L. James and Dinesh K. Pai. Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects. *ACM Trans. on Graphics*, 22(1):47–82, 2003.

[JP04] Doug L. James and Dinesh K. Pai. BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models. *ACM Trans. on Graphics*, 23(3):393–398, August 2004.

[JT05] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Transactions on Graphics*, 24(3):399–407, August 2005.

[KcM00] U. Kühnapfel, H. Çakmak, and H. Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24:671–682, 2000.

[KKR00] Jeonghwan Ko, Andrew J. Kurdila, and Othon K. Rediniotis. Divergence free bases and multiresolution methods for reduced-order flow modeling. *AIAA Journal*, 38(12):2219–2232, 2000.

[KKSD02] Laehyun Kim, Anna Kyrikou, Gaurav S. Sukhatme, and Mathieu Desbrun. Implicit-based Haptic Rendering Technique. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2002.

[KLM01] P. Krysl, S. Lall, and J. E. Marsden. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *Int. J. for Numerical Methods in Engineering*, 51:479–504, 2001.

[KMH+04] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Contact Handling for Deformable Point-Based Objects. In *Proc. of Vision, Modeling, Visualization 2004*, pages 339–346, 2004.

[KOLM02] Young J. Kim, Miguel A. Otaduy, Ming C. Lin, and Dinesh Manocha. Six-Degree-of Freedom Haptic Display Using Localized Contact Computations. In *The Tenth Symp. on Haptic Interfaces For Virtual Environment and Teleoperator Systems*, March 24-25 2002.

[KOLM03] Young J. Kim, Miguel A. Otaduy, Ming C. Lin, and Dinesh Manocha. Six degree-of-freedom haptic display using incremental and localized computations. *Presence-Teleoperators and Virtual Environments*, 12(3):277–295, 2003.

[Kuf04] James J. Kuffner. Effective sampling and distance metrics for 3d rigid body path planning. In *IEEE Int. Conf. on Robotics and Automation 2004*, April 2004.

[LB05] Ren-Cang Li and Zhaojun Bai. Structure preserving model reduction using a Krylov subspace projection formulation. *Comm. Math. Sci.*, 3(2):179–199, 2005.

[LCN99] J.C. Lombardo, M.P. Cani, and F. Neyret. Real-time Collision Detection for Virtual Surgery. In *Proc. of Comp. Animation 1999*, May 1999.

[LD07] S.D. Laycock and A.M. Day. A Survey of Haptic Rendering Techniques. *Computer Graphics Forum*, 26:50–65, 2007.

[LLVT03] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares. Efficient implementation of Marching Cubes' cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

[LMG02] S. Lall, J. E. Marsden, and S. Glavaski. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *Int. J. on Robust and Nonlinear Control*, 12:519–535, 2002.

[LS76] Wai-Hung Liu and Andrew H. Sherman. Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices. *SIAM Journal on Numerical Analysis*, 13(2):198–213, 1976.

[LS07] François Labelle and Jonathan Richard Shewchuk. Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angl es. In *Proc. of ACM SIGGRAPH 2007*, Aug 2007.

[Lum67] J. L. Lumley. The structure of inhomogeneous turbulence. In A.M.Yaglom and V.I.Tatarski, editors, *Atmospheric turbulence and wave propagation*, pages 166–178, 1967.

[MAC04] Damien Marchal, Fabrice Aubert, and Christophe Chaillou. Collision Between Deformable Objects Using Fast-Marching on Tetrahedral Models. In *Proc. of the Symp. on Comp. Animation 2004*, Aug 2004.

[MBTF03] Neil Molino, Robert Bridson, Joseph Teran, and Ron Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable*, pages 103–114, 2003.

[MDM+02] M. Müller, J. Dorsey, L. McMillian, R. Jagnow, and B. Cutler. Stable Real-Time Deformations. In *Proc. of the Symp. on Comp. Animation 2002*, pages 49–54, 2002.

[MG04] M. Müller and M. Gross. Interactive Virtual Materials. In *Proc. of Graphics Interface 2004*, pages 239–246, 2004.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless Deformations Based on Shape Matching. In *Proc. of ACM SIGGRAPH 2005*, pages 471–478, Aug 2005.

[MO06] C. Mendoza and C. O'Sullivan. Interruptible collision detection for deformable objects. *Computer & Graphics*, 30(2), 2006.

[MPT99] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *Proc. of ACM SIGGRAPH 99*, pages 401–408. ACM, 1999.

[MPT06] W.A. McNeely, K.D. Puterbaugh, and J.J. Troy. Voxel-Based 6-DOF Haptic Rendering Improvements. *Haptics-e*, 3(7), 2006.

[MT92] Dimitri Metaxas and Demetri Terzopoulos. Dynamic Deformation of Solid Primitives with Constraints. *Computer Graphics (Proc. of ACM SIGGRAPH 92)*, 26(2):309–312, 1992.

[MTG04] M. Müller, M. Teschner, and M. Gross. Physically-Based Simulation of Objects Represented by Surface Meshes. In *Proc. of Comp. Graphics Int. (CGI)*, pages 26–33, Crete, Grece, June, 16–19 2004.

[Nic76] R. E. Nickell. Nonlinear Dynamics by Mode Superposition. *Computer Methods in Applied Mechanics and Engineering*, 7:107–129, 1976.

[Nin02] Ning Hu and Vahe Poladian. Characterizing and Enhancing the Performance of Sound Synthesis Applications on Pentium III, Computer Architecture Final Course Project Report, http://www.cs.cmu.edu/ ninghu/papers/Report_CA02.pdf, 2002.

[Noo81] Ahmed K. Noor. Recent Advances in Reduction Methods for Nonlinear Problems. *Computers and Structures*, 13:31–44, 1981.

[NP80] Ahmed K. Noor and Jeanne M. Peters. Reduced Basis Technique for Nonlinear Analysis of Structures. *AIAA Journal*, 18(4):455–462, 1980.

[OG07] Miguel A. Otaduy and Markus Gross. Transparent Rendering of Tool Contact with Compliant Environments. In *Proc. of the World Haptics Conference*, pages 225–230, 2007.

[OH99] J.F. O'Brien and J.K. Hodgins. Graphical Modeling and Animation of Brittle Fracture. In *Proc. of ACM SIGGRAPH 99*, pages 111–120, 1999.

[OL03] Miguel A. Otaduy and Ming C. Lin. Sensation Preserving Simplification for Haptic Rendering. In *Proc. of ACM SIGGRAPH 2003*, pages 543–553. ACM, 2003.

[OL05] Miguel A. Otaduy and Ming C. Lin. Stable and Responsive Six-Degree-of-Freedom Haptic Manipulation Using Implicit Integration. In *Proc. of the World Haptics Conference*, pages 247–256, 2005.

[OL06] Miguel A. Otaduy and Ming C. Lin. A Modular Haptic Rendering Algorithm for Stable and Transparent 6-DoF Manipulation. *IEEE Trans. on Robotics*, 22(4):751–762, 2006.

[OSG02] J. F. O'Brien, C. Shen, and C. M. Gatchalian. Synthesizing Sounds from Rigid-Body Simulations. In *Proc. of the Symp. on Comp. Animation 2002*, pages 175–181, 2002.

[Ota04] Miguel A. Otaduy. *6-DoF Haptic Rendering Using Contact Levels of Detail and Haptic Textures*. PhD thesis, Department of Comp. Science, University of North Carolina at Chapel Hill, 2004.

[Pai02] Dinesh Pai. Strands: Interactive simulation of thin solids using Cosserat models. *Computer Graphics Forum*, 21(3):347–352, 2002.

[Par01] Rick Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann, 2001.

[PDA01] G. Picinbono, H. Delingette, and N. Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. In *IEEE Int. Conf. on Robotics and Automation 2001*, May 2001.

[PPG04] M. Pauly, D. K. Pai, and L. Guibas. Quasi-Rigid Objects in Contact. In *Proc. of the Symp. on Comp. Animation 2004*, 2004.

[PW89] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics (Proc. of ACM SIGGRAPH 89)*, 23(3):215–222, July 1989.

[PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface Elements as Rendering Primitives. In *Proc. of ACM SIGGRAPH 2000*, pages 335–342, 2000.

[RJ07] Alec R. Rivers and Doug L. James. FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. In *Proc. of ACM SIGGRAPH 2007*, Aug 2007.

[RKK97] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The Haptic Display of Complex Graphical Environments. In Turner Whitted, editor, *Proc. of ACM SIGGRAPH 97*, pages 345–352. Addison Wesley, August 1997.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 343–352, July 2000.

[RPP+01] M. Renz, C. Preusche, M. Pötke, H.-P. Kriegel, and G. Hirzinger. Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *Proc. of Eurohaptics*, pages 149–154, 2001.

[Sen] Sensable Technologies, Inc. Phantom haptic interfaces, http://www.sensable.com.

[SF92] Mikio Shinya and Alain Fournier. Stochastic motion - Motion under the influence of wind. *Comp. Graphics Forum*, pages 119–128, 1992.

[SGG+06a] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha. Fast Proximity Computation Among Deformable Models using Discrete Voronoi Diagrams. In *Proc. of ACM SIGGRAPH 2006*, 2006.

[SGG+06b] Avneesh Sud, Naga Govindaraju, Russell Gayle, , and Dinesh Manocha. Interactive 3D Distance Field Computation using Linear Factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games (I3D)*, 2006.

[Sha90] Ahmed A. Shabana. *Theory of Vibration, Volume II: Discrete and Continuous Systems*. Springer–Verlag, New York, NY, 1990.

[Sha05] Ahmed A. Shabana. *Dynamics of Multibody Systems*. Cambridge Univ. Press, New York, NY, 2005.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In *Proc. of ACM SIGGRAPH 1985*, pages 245–254, 1985.

[SL06] H. Hoppe S. Lefebvre. Perfect Spatial Hashing. In *Proc. of ACM SIGGRAPH 2006*, Aug 2006.

[SM01] S.Fisher and M.C.Lin. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 330–336, 2001.

[SOM04] Avneesh Sud, Miguel Otaduy, and Dinesh Manocha. DiFi: Fast 3D Distance Field Computation Using Graphics Hardware. *Comp. Graphics Forum*, 23(3):557–556, 2004.

[SOS04] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*, pages 896–904. ACM Press, August 2004.

[SPG03] Christian Sigg, Ronny Peikert, and Markus Gross. Signed Distance Transform using Graphics Hardware. In *Proceedings of IEEE Visualization Conference*, pages 83–90, 2003.

[Sta97] J. Stam. Stochastic Dynamics: Simulating the Effects of Turbulence on Flexible Structures. *Comp. Graphics Forum*, 16(3), 1997.

[SV97] S. Salcudean and T. Vlaar. On the emulation of stiff walls and static friction with a magnetically levitated input-output device. *ASME Journal of Dynamics, Measurement and Control*, 119:127–132, 1997.

[TBHF03] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 68–74, August 2003.

[TLP06] A. Treuille, A. Lewis, and Z. Popović. Model Reduction for Real-time Fluids. In *Proc. of ACM SIGGRAPH 2006*, 2006.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically Deformable Models. *Computer Graphics (Proc. of ACM SIGGRAPH 87)*, 21(4):205–214, 1987.

[TSIF05] Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. Robust Quasistatic Finite Elements and Flesh Simulation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 181–190, July 2005.

[Tur92] Greg Turk. Re-Tiling Polygonal Surfaces. *Computer Graphics (Proc. of ACM SIGGRAPH 92)*, 26(2):55–64, 1992.

[Tur01] Greg Turk. Texture Synthesis on Surfaces. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 347–354, August 2001.

[TV77] A.N. Tikhonov and V.A.Arsenin. *Solution of Ill-posed Problems*. Winston & Sons, 1977.

[TW88] Demetri Terzopoulos and Andrew Witkin. Physically Based Models with Rigid and Deformable Components. *IEEE Comp. Graphics & Applications*, 8(6):41–51, 1988.

[WH94] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278, July 1994.

[WM03] Ming Wan and William A. McNeely. Quasi-Static Approximation for 6 Degrees-of-Freedom Haptic Rendering. In *Proc. of IEEE Visualization 2003*, pages 257–262, 2003.

[WRF+96] Mark William, Scott Randolph, Mark Finch, James Van Verth, and Russell M. Taylor II. Adding Force Feedback to Graphics Systems: Issues and Solutions. In *Proc. of ACM SIGGRAPH 96*, pages 447–452, August 1996.

[Wri02] Peter Wriggers. *Computational Contact Mechanics*. John Wiley & Sons, Ltd., 2002.

[WW90] Andrew Witkin and William Welch. Fast Animation and Control of Nonrigid Structures. *Computer Graphics (Proc. of ACM SIGGRAPH 90)*, 24(4):243–252, August 1990.

[ZC00] Yan Zhuang and John Canny. Haptic Interaction with Global Deformations. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, San Francisco, USA, 2000.

[ZS95] C.B. Zilles and J.K. Salisbury. A Constraint-based God-object Method for Haptics Display. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 146–151. IEEE, 1995.