Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2011)
J. Bender, K. Erleben, and E. Galin (Editors)

Precomputed Shape Database for Real-Time Physically-Based Simulation

Loeiz Glondu[1], Benoit Legouis[1], Maud Marchal[2] and Georges Dumont[1]

[1]ENS Cachan, Antenne de Bretagne, France
[2]INSA Rennes, France

Abstract

Adding dynamically physical properties to virtual objects can not generally be handled in real-time during a simulation. In this paper, we propose a method for handling the real-time physical simulations of arbitrary objects that are represented by their surface mesh. Our method is based on a database in which physical data are stored for a wide variety of objects. When a query object needs to be physically simulated in the virtual world, a similarity search is performed in the database and the associate physical data are then extracted. We propose and compare three different similarity search methods that fit with our real-time needs. We demonstrate our approach for the simulations of different physical phenomena such as fracture or deformations. Our results show that our method has a great potential for the physical simulation of objects represented by their surface meshes in interactive applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometry and Object Modeling—Physically based modeling I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

1. Introduction

Physically-based simulations are now commonplace in the computer graphics community, providing powerful tools for computer animation, games or Virtual Reality. It is currently possible to synthesize in real-time the motion of rigid and deformable bodies as well as fluids. For each simulation, different physical data are used, such as modal analysis or stiffness matrices for deformable bodies and inertia information for rigid bodies. These physical data are usually precomputed for each object and used in real-time simulations. However, there are various computer animations where physical data are not available. First, a lot of virtual scenes contains objects that are not systematically associated with physical properties but only represented by their surface meshes. It allows to accelerate the animations or to simplify their use. Consequently, the virtual worlds lack of physical behaviors and physical data that need to be precomputed can not be added at interactive rate. Second, there are physical simulations where new objects can be dynamically added to the virtual world. For example, it is the case in scenarios where some objects are fracturing: new objects with unpredictable geometry are created (see Figure 1). These new objects are not always associated with any physical mesh, and their behaviors are no more physically realistic. In both cases, the addition of new physical data in real-time is challenging but necessary for interactive applications that need stable frame rates.

In this paper, we propose a new approach for adding new physical data to arbitrary objects represented only by their surface meshes. Our contributions are :

- **A method for handling the physical simulation of arbitrary virtual objects represented by their surface meshes**. Precomputed physical data such as modes of deformation are associated to any object of a virtual world, providing in real-time new physical characteristics for computer animation.
- **A methodology for creating a database containing various physical data**. Various types of physical properties can be recorded. Our approach can thus be used through different scenarios involving deformable or rigid bodies.
- **The adaptation and a comparison of three similarity search methods for real-time physically based simulations**. From existing work on similarity search, we propose three descriptors for surface meshes that provide

DOI: 10.2312/PE/vriphys/vriphys11/047-054

a good trade-off between the accuracy and efficiency of classical methods.

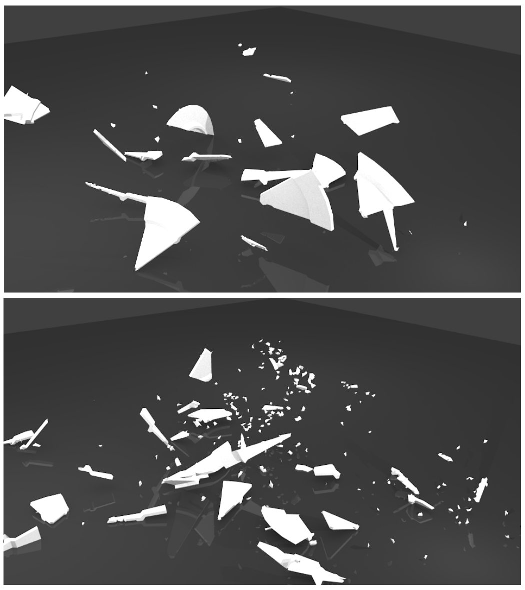
Figure 1: *Application of the precomputed shapes method to a plate which is broken through a procedural fracturing process. Each fragment is associated with collision detection and inertia data from the database. A rigid body simulation can then be run.*

The paper is organized as follows. In section 2, we present related work on similarity search in 3-D object databases. Section 3 presents the database and the process of the similarity search. Section 4 details the different similarity descriptors and their adaptation to real-time physical simulations. Section 6 shows the results of our approach through different scenarios.

2. Related work

Offline precomputation of physical data is common in physical simulations. Computing the physical data of an object necessitates to know its geometry and some of its mechanical properties such as its density or its elastic properties. A stiffness matrix [MG04] or modes of deformation [Bar07] from a modal analysis are good examples of precomputed physical data used for real-time simulation of deformations (we refer the reader to [NMK*06] for a survey on physically-based deformation techniques in computer graphics). However, in the context of interactive applications, the physical data cannot always be precomputed at run-time, especially when new objects are dynamically added to the scene. A solution to this problem is to store precomputed data into a database, and use them on objects for which no physical information is provided. In [ZJ10], the authors use a database containing "sound proxies" in order to accelerate the sound synthesis of shattering bodies. They propose a method to find ellipsoid that match with their fragments. In order to extend this principle to have databases containing any variety of shape, it is possible to leverage the widely studied field of similarity search in databases of 3-D objects.

Similarity search in database of 3-D objects retrieval purpose is to find objects of a database that are similar to an input query object (see [BKS*05] for a survey on the topic). It is often performed by extracting features from the geometry of the query object and comparing these features to the entry of the database. The features extracted from the mesh of the body is called *feature vector* or *descriptor* of the object. The descriptors extracted from the object geometry can be built from their surface mesh, volumetric mesh, or from 2-D images of the object [HKSV02]. In the literature, various kind of features are extracted from the objects, such as boxes [PMN*00], spherical harmonics [VSR01], Reeb graphs [HSKK01], moments of inertia [BKS*05] or statistical information extracted from voxel representations [VS01] or surface distribution [OFCD02].

The descriptors can be classified on their efficiency (how long it takes to build it, and how long it takes to find the best match based on the similarity) and on their accuracy (how the defined similarity is relevant w.r.t. the targeted application). There is currently no method capable of capturing the local geometry features of the objects and performing a similarity search at interactive rates. In our context, efficiency is a crucial criterion, as we target interactive applications. We adapted voxel-based representations and moment-based methods to our needs to provide descriptors that propose a good trade-off between efficiency and accuracy.

3. Precomputed Shape Database

This section presents how the database is built offline in a first part, and how the online similarity search is performed is this database in a second part.

3.1. Creating the Database

We propose a process of creation of the database in five steps: (1) the generation of the surface mesh, (2) the computation of their volumetric meshes, (3) a normalization step, (4) the computation of the descriptors, and (5) the addition of physical data. These steps are detailed along this section.

Choice and Generation of Surface Meshes The choice of the type of shapes that are stored in the database mainly depends on the targeted application. We arbitrarily selected the six main generic shapes that are shown in Figure 3.

From each main shape, we generate a set of surface meshes by scaling the initial shape along two or three axis depending on their symmetrical properties. In our example, about 4,500 surface meshes have been created from the six

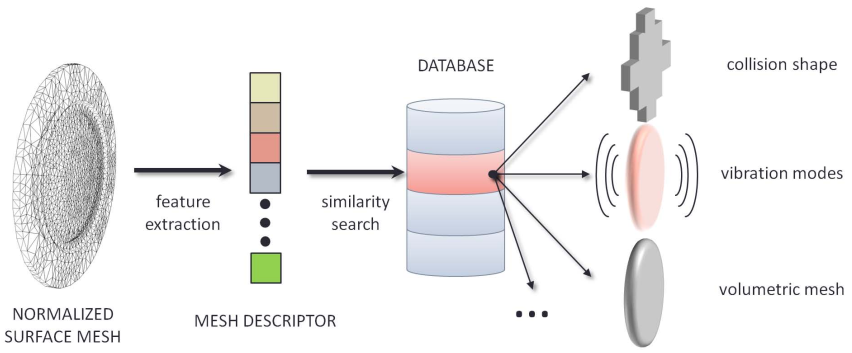

Figure 2: *Overview of the similarity search process. Features are extracted from the normalized input surface mesh to obtain the mesh descriptor. This descriptor is compared to the descriptor of each database entry to find the best match. Each database entry contains physical data such as a collision shape for collision detection module, the lowest vibrations modes of the body or a volumetric mesh.*

main shapes. Each surface mesh follows the four following treatments to add an entry in the database.

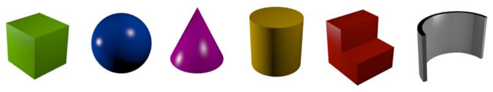

Figure 3: *The six main shapes used to generate our database. Each main shape generates hundreds of derived shapes by changing their size parameters.*

Computation of the Volumetric Mesh In order to compute the volume, mass and inertia matrices of the surface mesh, we first compute the volumetric mesh from the surface mesh. In that purpose, we apply a conforming Delaunay tetrahedralization using TetGen software [Tet] to the surface mesh (TetGen is also able to rearrange the surface points to reinforce the quality of the mesh, or to apply a tetrahedron shape constraints). Once the tetrahedral mesh is built, we compute its center of mass, volume, and principal rotation axis (from an eigen-decomposition of its inertia matrix).

Normalization of the Shape We normalize the surface mesh so that the extracted descriptor are translation, rotation and scaling-invariant (see Figure 4). We transform the input surface mesh to obtain a volume of $1\,m^3$, so that its center of mass coincides with the origin of the coordinate system, and so that its principal rotation axis are aligned with the axis of the coordinate system. We chose to align the x-axis with the principal rotation axis that has the lowest inertia, and to align the y-axis with the second principal rotation axis (the direction of the third axis is arbitrarily determined on whether a left-handed or right-handed coordinate system is used).

Creation of the Descriptor Once the surface mesh is normalized, we extract a descriptor $\mathbf{d} \in \mathbb{R}^n$. The size n and value

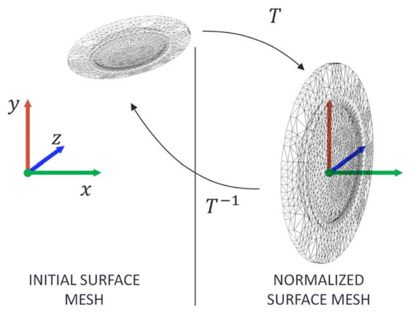

Figure 4: *Normalization of a surface mesh. After normalization, the center of mass coincides with the origin, the principal rotation axes coincide with the axes of the coordinate system, and the volume of the mesh is equal to 1. The affine transformation applied for the normalization is stored in a matrix $T \in \mathbb{R}^{4\times4}$.*

of the descriptor depends on the type of descriptor used. The extraction of the descriptor is explained in section 4. When the descriptor is created, a new entry labeled with **d** is created into the database.

Adding Physical Data To each entry of the database labeled by the value of a mesh descriptor **d**, we associate as many physical data as needed by the simulation algorithms. For example, each entry can contain one or more of the follow data (see also Figure 2):

- A normalized tetrahedral volumetric mesh, that represents the shape of the object.
- The n first modes of deformation (computed by performing a modal analysis with either LAPACK [LAP] or ARPACK [ARP] software on the volumetric mesh). These deformation modes of a body enable to efficiently

compute its small deformations due to input impulses [OSG02].
- Precomputed radiation data for sound generation [GD04, JBP06]. These models can be used in association with the deformation modes to efficiently generate the sound of the objects.
- A collision shape structure: a compound shape built *e.g.* with spheres, cube and cylinders for rigid body simulators.

3.2. Searching into the Database

When a new object represented by its surface mesh is added to the scene, a similarity search is performed in the database to retrieve the needed physical data for simulation. In other words, given a query input mesh, we want to find in real-time the entry into the database which is the most similar to the input mesh. To perform the similarity search, the input mesh is first normalized as explained in section 3.1 to ensure translation, rotation and scale invariances. A descriptor $\mathbf{d}_{in}$ is then extracted from the normalized mesh. Finally, we select the entry e_i associated with its descriptor $\mathbf{d}_i$ into the database for which the similarity value $v(\mathbf{d}_{in}, \mathbf{d}_i)$ is the lowest (see Figure 2):

$$e_i = \underset{i}{\operatorname{argmin}}\left(v(\mathbf{d}_{in}, \mathbf{d}_i)\right) \tag{1}$$

In practice, the evaluation of $v(\mathbf{d}_{in}, \mathbf{d}_i)$ depends of the descriptor type and dimensionality. Moreover, it is often not necessary to check all the entries of the database to find the best match. Depending on the descriptor used, the database can be organized to optimize the similarity search, which are interesting properties w.r.t. real-time needs. These aspects are discussed in the following section.

4. Mesh Descriptors

We detail in this section the different descriptors used for the search request in the object database. In our context, an object descriptor $\mathbf{d} \in \mathbb{R}^n$ is a scalar vector of dimension n that is build from its surface mesh. The dimension as well as the content of the descriptor depends on the features extracted from the initial mesh (see Section 2 for more details). The real-time requirements impose several constraints on the choice of the descriptor. First, the descriptor must be efficiently built. Moreover, the dimension n of the descriptor should not be excessive, in order to avoid long search time w.r.t. the real-time update frequency. Finally, we preferred descriptors that enable to build partial ordered sets, to have more efficient search algorithms. With regards to these constraints, we retained three descriptors: a moment-based descriptor, a voxel-based descriptor and an improved voxel-based descriptor. Each descriptor is discussed along this section.

4.1. Moment-based Descriptor

Moment-based descriptor relies on the mass distribution of the material around axis. We compute the moment of inertia of the object around specific axes (we chose the three orthogonal axes aligned with the system of coordinates). The moment of inertia I around an axis $\mathbf{u}$ is computed by integrating the mass of the object around the axis $\mathbf{u}$ and over the volume V of the object:

$$I = \int_V dist(\mathbf{p}, \mathbf{u})^2 \cdot \rho(\mathbf{p})\, dV(\mathbf{p}) \tag{2}$$

where $\mathbf{p}$ is a position into the volume, $\rho(\mathbf{p})$ is the material density at $\mathbf{p}$ and $dist(\mathbf{p}, \mathbf{u})$ is the distance between position $\mathbf{p}$ and axis $\mathbf{u}$. In our context, objects are defined with discrete geometry, and the moment of inertia is computed considering discrete positions $\mathbf{p}_i$ and masses m_i:

$$I = \sum_{i<n} dist(\mathbf{p}_i, \mathbf{u})^2 m_i \tag{3}$$

where n is the number of considered point-masses. In the case of surface meshes, we spread the mass of the object into its mesh nodes for the moment of inertia computation. If a tetrahedral volumetric mesh is available, it is possible to compute the moment of inertia using centers of tetrahedral elements as $\mathbf{p}_i$ and we use the volume of the element and its density to compute the associated m_i in equation (3).

We retain in the descriptor $\mathbf{d}$ the three moments of inertia computed around x-axis, y-axis and z-axis, sorting the values ascendantly. The descriptors are the sorted in the database, enabling dichotomous efficient search.

Similarity Computation The similarity v between two moment descriptors $\mathbf{d}_1$ and $\mathbf{d}_2$ is computed using the square of the Euclidean distance between the two descriptor vectors:

$$v = ||\mathbf{d}_2 - \mathbf{d}_1||^2 \tag{4}$$

4.2. Voxel-based Descriptor

Our version of voxel-based descriptors relies on a uniform grid that stores the spatial distribution of the surface of the object. We center a cubic uniform grid at the object center of mass, and size the grid so that its side is equal to the largest side of the bounding box of the input object. Each cell of the grid is marked with 0 if it contains no node of the object mesh, and 1 otherwise (see Figure 5(b)). The number of cells in the grid is a parameter of the descriptor that has consequences on the search results. The voxel-based descriptor $\mathbf{d}$ is built by writing the 3-D grid in a linear way.

Similarity Computation The similarity v between two voxel-based descriptors is a value representing the number of cells that have a different value:

$$v = \frac{1}{n_{filled}} \sum_{i \in dim(\mathbf{d})} dif(d_{1i}, d_{2i}) \tag{5}$$

where n_{filled} is the total number of non empty cells in both $\mathbf{d}_1$ and $\mathbf{d}_2$, and $dif(a,b)$ is 0 if a and b are equals, 1 otherwise. The normalization of the distance using $1/n_{filled}$ ensures that $0 \leq v \leq 1$, and that the objects that contain a large number of non empty cells do not reduce artificially the distance between the descriptors.

4.3. Improved Voxel-based Descriptor

We propose a modified version of the voxel-based descriptor that avoids most non-identical meshes to have the same descriptor. The improved voxel-based descriptor contains the information of the voxel-based descriptor, but we add to each cell three scalars that represent the location of the center of mass of all the nodes that belong to this cell. This position gives an information on the distribution of the points inside the cell (see Figure 5(c)).

Similarity Computation The similarity v between two improved voxel-based descriptors is computed using the Euclidean distance between the centers of mass of the cells. This distance is normalized using the length l_{diag} of the diagonal of the cubic cell. If one cell of a descriptor is empty while the same cell of the other descriptor is not, the distance is set to 1:

$$v = \frac{1}{n_{filled} + n_{eq}} \cdot \sum_{i \in dim(\mathbf{d})} \left(dif(d_{1i}, d_{2i}) + (1 - dif(d_{1i}, d_{2i})) \times \frac{\|\mathbf{c}_{1i} - \mathbf{c}_{2i}\|}{l_{diag}} \right) \tag{6}$$

where n_{eq} is the number of common cells between $\mathbf{d}_1$ and $\mathbf{d}_2$ that are both equal to 1, $\mathbf{c}_i$ is the position of the computed center of mass of cell i.

5. Adaptation to Physical Simulation

Scaling the Physical Data The physical data contained in the database are all normalized. In order to use them for physical simulation, there are two possibilities: (1) make a copy of the physical data and transform it so that it fits with the query object, or (2) transform the input data of the physical simulation so that it fits with the normalized object. The solution (1) consumes more memory, but it can be applied for most physical data (such as a stiffness matrix or deformation modes). Solution (2) can be more efficient, but can be applied only if the physical data properties scales linearly with a scale of the geometry of the object.

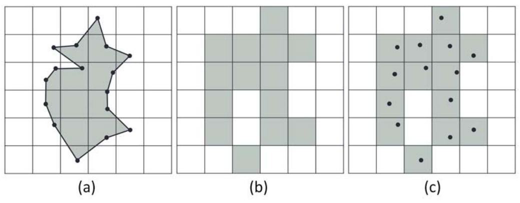

Figure 5: *Voxel-based descriptors used in our system.* ***(a)****: normalized surface mesh and its bounding grid. The center of mass of the mesh coincides with the center of the grid.* ***(b)****: Simple voxel-based descriptor that stores the cells of the grid containing at least one node of the surface mesh.* ***(c)****: Improved voxel-based descriptor that stores additionally the position of the center of mass of the nodes belonging to the cell.*

For example, the deformation modes, stiffness matrix and the collision data can be transformed using the inverse transformation T^{-1} of the normalization (see Figure 4). The geometric scale of the body is used to update the physical values (see [ZJ10] for a derivation of how to scale stiffness matrix and mode of deformation). The volumetric mesh can be used to find a correspondence between a position $\mathbf{p} \in \mathbb{R}^3$ and the closest vertex of the mesh to this position (for collision and force application purposes). In this case, the mesh is not scaled nor copied, but the input position $\mathbf{p}$ is transformed into the normalized coordinates using $\mathbf{p}' = T\mathbf{p}$. Then, the new position $\mathbf{p}'$ is used to find the closest vertex of the mesh.

Linking the Physical Data with the Virtual Objects When simulating the deformation of the objects, we need to propagate the deformation stored into the physical data to the initial surface mesh. To do so, we express each Degree Of Freedom (DOF) f_i of the initial surface mesh from a weighted sum of the degree of freedom of the physical mesh:

$$f_i = \mathbf{b}.\mathbf{q} \tag{7}$$

where $\mathbf{b}$ is a vector of weights (we impose $\sum_{i \in dim(\mathbf{q})} b_i = 1$), and $\mathbf{q}$ is the physical state of the body. In practice, we choose to find for each node of the surface mesh the element of the volumetric mesh that contains it. Then, we link the node of the surface mesh and the node of the element using barycentric coordinates (see Figure 6).

Database Search Optimization Depending on the descriptor used, several optimizations can be applied to accelerate the database requests. For voxel-based descriptors, we set up an optimization based on the number of empty cells. We gather the descriptors that have the same number of empty cells. When a request is performed, the number of empty

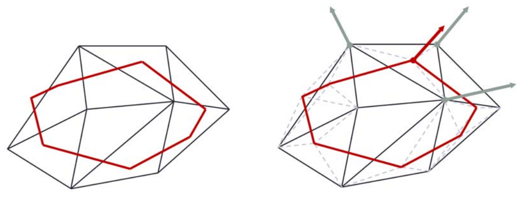

Figure 6: *Link between the surface mesh and physical data. The red mesh represents the initial object surface mesh. The black mesh is the best match from the database, transformed to fit with the surface mesh. Right: the surface mesh DOFs are linked to the physical mesh DOFs using the nodes of the element that contains the node of the surface mesh (represented with gray dotted lines). The red arrow is the displacement of the surface mesh node computed from the displacement of the physical mesh.*

cells of the input descriptor is used to access directly the list of descriptors that have the same number of empty cells. This optimization enables to save computation time for each request. However, comparing the query object with the objects that have exactly the same number of empty cells in their descriptor is too restrictive, as we can miss the best entry in terms of equation (1). In practice, we take all the descriptors in the descriptors that have a number of empty cells n_e in the range $[n_e - r, n_e + r]$, where r is a manually set parameter (setting r to 6 in our test scenarios led to get always the best entry).

6. Results

Summary of the Method The overview of our method is presented in algorithm 1. This algorithm describes how the physical data is retrieved from the input surface mesh $\mathcal{S}$ using the database noted $\mathcal{D}$.

Algorithm 1 Find the best physical data from a surface mesh

```
Require: $\mathcal{S}$: input surface mesh
Require: $\mathcal{D}$: shape database
 1: $T$ = normalize_transform($\mathcal{S}$)
 2: $\mathcal{S}'$ = transform($\mathcal{S}$, $T$)
 3: $\mathbf{d}$ = descriptor_from($\mathcal{S}'$)
 4: $e_i$ = best_entry($\mathbf{d}$, $\mathcal{D}$)          // eqn (1)
 5: $\mathcal{C}$ = collision_volume($e_i$)
 6: $\mathcal{M}$ = mode_deformation($e_i$)
 7: $\mathcal{C}$ = transform($\mathcal{C}$, $T^{-1}$)
 8: $\mathcal{M}$ = transform($\mathcal{M}$, $T^{-1}$)
 9: scale $\mathcal{M}$
10: link $\mathcal{M}$ and $\mathcal{C}$ to $\mathcal{S}$
```

Configuration Our simulation method has been implemented in C++ on a laptop with 4 GB of RAM, and an Intel®Core$^{\text{TM}}$2 Extreme. The size of the grid for both voxel-based and improved voxel-based descriptors has been experimentally set to $10 \times 10 \times 10$.

Scenarios and Computation Time Performances Figures 1 and 8 show a scenario where a plate is broken, and each fragment is assigned with a physical data. We show an example of scenario where rigid bodies simulation is augmented to incorporate physically-based deformations computed with modal analysis in Figure 10. The improved voxel descriptor gives the best results thanks to the mass distribution information within each cell of their grid. On the opposite side, the moment-based descriptor gives less relevant results, due to a less efficient separation of the information of the mass distribution, and a lower dimensionality.

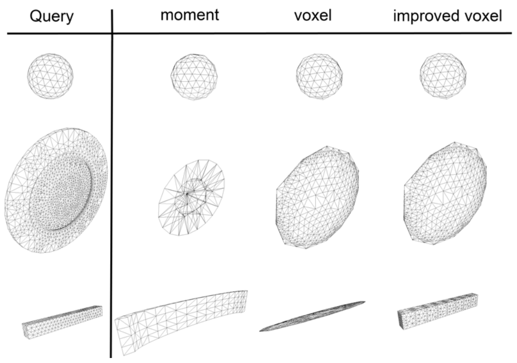

Figure 7: *Comparison of similarity search methods. The moment-based method gives more naive results, while the improved voxel-based method finds visually similar objects for the ball, the plate and the beam.*

The timings for scenario in Figure 9 shows that our solution is satisfying for interactive application. Indeed, the optimized version of the improved voxel based descriptor need a total of 4ms of processing time per fragment in average (the similarity search complexity is independent of the complexity of the input mesh).

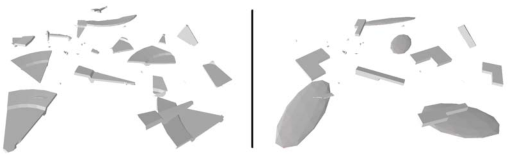

Figure 8: *Fragments associated to physical meshes in a fracture scenario. Left: surface meshes representing the fragments. Right: physical meshes associated to the same fragments.*

7. Discussion

Physical Realism The main limitation of our approach resides on the physical correctness that is achieved by using an

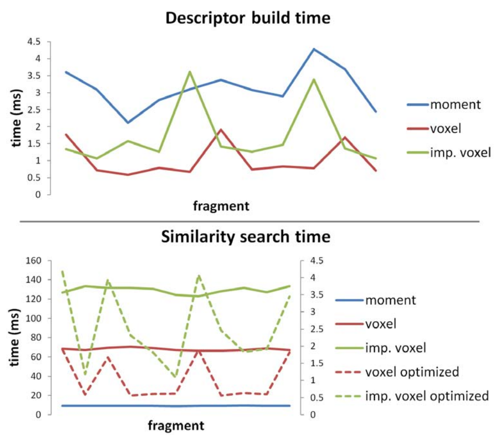

Figure 9: *Computation times for descriptor build and similarity search measured from scenario of Figure 8. The solid line are relative to the left axis, while the dotted lines are relative to the right axis.*

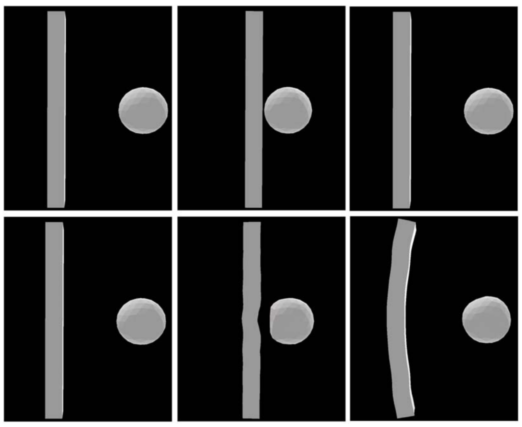

Figure 10: *Enriching a rigid body simulation with physically-based deformable features. Top: a rigid rod is thrown against a rigid sphere. Bottom: The same scenario, but the two bodies have retrieved from the database modes of deformation, allowing to add physically based deformation due to the contact force.*

approximated geometry. Indeed, the precomputed database contains a limited number of shapes, and complex concave objects have a small chance to be associated with acceptable data. It would be however possible to enrich the database with the objects during a specific scenario in an offline mode. This would result in a specific scenario-based database, and limit the physical error. Future work will concern a measurement of the physical correctness of applying approximated geometry on specific scenarios. Also, it would be necessary to establish a metric to measure the performances of the similarity search algorithm.

Physical Data and Scaling As highlighted in section 5, the physical data of the database should be scaled to fit with the size of the non-normalized query object. Even if most of physical data can be scaled with an homogeneous scale of the geometry of the body, our method can not be applied with precomputed data that can not scale with the geometry of the body.

Database Management Currently, the database is entirely loaded into the RAM in order to provide a quick access to the data. However, increasing the number of objects in the database and/or the quantity of physical data would consume too much memory to be stored in the RAM. A solution could be to store only the objects descriptor in the RAM, and access the physical data from a massive storage device only when it is needed.

Applications of our Method We shown in our results a few scenarios where our method is applied. The precomputed database method can actually be use in a wide range of interactive applications, such as:

- Sound synthesis: generating physically based sound can be computationally costly. Precomputing vibration modes or other sound-based physical data can greatly reduce the computation time as proposed in [ZJ10].
- Enriching existing applications with physics: many interactive application do not integrate physical simulation. Our method is a mean of adding physical simulation system at low cost on existing non physical virtual worlds.
- Collision detection: collision detection between complex objects is often the bottleneck of interactive simulations. It would be possible however to compute optimized collision detection structures (such as hierarchies of polytopes based bounding volumes [KHM*98]) to remove this bottleneck.
- Haptic rendering: Interactions that include haptic rendering must run at high frequencies. Our method could be used to allow physically based interaction in various scenarios that include haptic rendering: the objects that are close to the proxy can be augmented with physical data at run time, and improve the interaction.

8. Conclusion

We have presented a method for real-time physically-based simulation of objects for which no physical data have been defined. Our method relies on a carefully generated database that stores all needed physical data, for a wide set of shapes. Each entry of the database contains a set of physical data for physical simulation. We also presented and compared three similarity search, and demonstrated their efficiency through scenarios of interactive physical simulation. We finally introduced an approach to scale and link the physical data stored in the database to the virtual objects. Our results show that the precomputed shape database proposes an interesting alternative solution for physical simulation in interactive applications.

References

[ARP] ARPACK:. http://www.caam.rice.edu/software/ARPACK/. 3

[Bar07] BARBIČ J.: *Real-time reduced large-deformation models and distributed contact for computer graphics and haptics*. PhD thesis, 2007. AAI3279452. 2

[BKS*05] BUSTOS B., KEIM D. A., SAUPE D., SCHRECK T., VRANIĆ D. V.: Feature-based similarity search in 3d object databases. *ACM Computer Survey 37* (December 2005), 345–387. 2

[GD04] GUMEROV N., DURAISWAMI R.: *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Elsevier, 2004. 4

[HKSV02] HECZKO M., KEIM D. A., SAUPE D., VRANIC D. V.: A method for similarity search of 3d objects. *Datenbank-spektrum 2* (2002), 54–63. 2

[HSKK01] HILAGA M., SHINAGAWA Y., KOHMURA T., KUNII T. L.: Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of SIGGRAPH* (2001), pp. 203–212. 2

[JBP06] JAMES D. L., BARBIČ J., PAI D. K.: Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. In *Proceedings of SIGGRAPH* (2006). 4

[KHM*98] KLOSOWSKI J., HELD M., MITCHELL J., SOWIZRAL H., ZIKAN: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics* (1998), 21–36. 7

[LAP] LAPACK:. http://www.netlib.org/lapack/. 3

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proceedings of Graphics Interface* (2004), pp. 239–246. 2

[NMK*06] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4 (2006), 1–24. 2

[OFCD02] OSADA R., FUNKHOUSER T., CHAZELLE B., DOBKIN D.: Shape distributions. *ACM Transaction on Graphics 21* (2002), 807–832. 2

[OSG02] O'BRIEN J. F., SHEN C., GATCHALIAN C. M.: Synthesizing sounds from rigid-body simulations. In *Proceedings of SIGGRAPH* (July 2002), ACM Press, pp. 175–181. 4

[PMN*00] PAQUET E., MURCHING A., NAVEEN T., TABATABAI A., RIOUX M.: Description of shape information for 2-d and 3-d objects. *Signal Processing: Image Communication 16* (2000), 103–122. 2

[Tet] TETGEN:. http://tetgen.berlios.de/. 3

[VS01] VRANIC D. V., SAUPE D.: 3d shape descriptor based on 3d fourier transform. In *Proceedings of EURASIP* (2001), pp. 271–274. 2

[VSR01] VRANIC D. V., SAUPE D., RICHTER J.: Tools for 3d-object retrieval: Karhunen-loeve transform and spherical harmonics. In *IEEE MMSP* (2001), pp. 293–298. 2

[ZJ10] ZHENG C., JAMES D. L.: Rigid-body fracture sound with precomputed soundbanks. In *Proceedings of SIGGRAPH* (July 2010), vol. 29. 2, 5, 7