Eurographics Italian Chapter Conference (2010)
E. Puppo, A. Brogni, and L. De Floriani (Editors)

An Application of Multiresolution Massive Surface Representations to the Simulation of Asteroid Missions

Giovanni Pintore[1], Roberto Combet[1], Enrico Gobbetti[1], Fabio Marton[1], and Russell Turner[2]

[1] Visual Computing Group, CRS4, Pula, Italy
[2] Johns Hopkins University Applied Physics Laboratory, Space Department, Laurel, Maryland, USA

Abstract

We report on a real-time application supporting fast, realistic real-time rendering of asteroid datasets, as well as collision detection and response between the asteroid and prototype robotic surface exploration vehicles. The system organizes the asteroid surface into a two-level multiresolution structure, which embeds a fine-grained per-patch spatial index within a coarse-grained patch-based structure. The coarse-grained structure, maintained out-of-core, is used for fast batched I/O and GPU accelerated rendering, while the per-patch fine-grained structure is used to accelerate raycasting and collision queries. The resulting system has been tested with a simple robot lander and surface exploration simulator. The system models gravity using mass particles uniformly distributed within the asteroid bodies. Real-time performance is achieved on a commodity platform with giga triangle representations of asteroids 25143 Itokawa and 433 Eros.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.6]: Methodology and Techniques—Computer Graphics [I.3.7]: Three-dimensional graphics and realism—Computer Graphics [I.3.8]: Applications—

1. Introduction

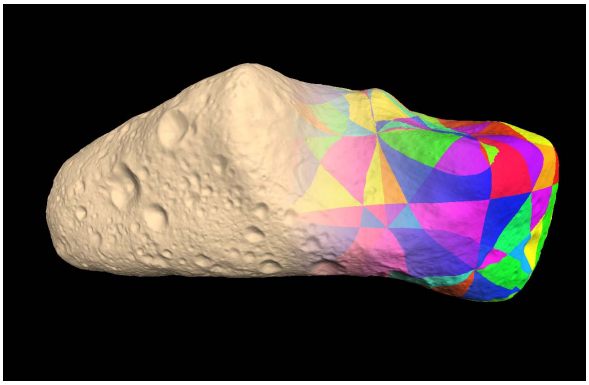

Figure 1: ***View-dependent rendering of 1.5Gtriangles Eros model.*** *The blended right side of the picture shows the adaptive mesh structure with a different color for each patch. Each patch corresponds to about 16K triangles.*

While the majority of asteroids orbit the sun in a region between Mars and Jupiter, a small number known as Near-Earth asteroids (NEAs) have orbits that are close to Earth's. Because of the potential hazard they pose to Earth as well as their accesibility, NEAs are compelling targets for robotic, and eventually human, exploration. However, only two of them have been landed on by a spacecraft: 433 Eros, by NASA/JHUAPL's Near Earth Asteroid Rendezvous (NEAR) probe and 25143 Itokawa, by the JAXA Hayabusa mission. Due to their extremely low-gravity environment (Itokawa's equatorial surface gravity is about $0.0001\frac{m}{s^2}$) and their complex shape topology, the technical problems of landing on asteroids can in many ways be more challenging than that of landing on planets or moons. It is therefore important to conduct realistic physics-based computer simulations during the spacecraft design process in order to study its likely performance in proximity to and on the surface of an asteroid. The ability to view and interact with such a simulation in real-time is particularly useful, as it can allow spacecraft designers to gain insight and an intuitive feel for such an exotic environment. In this context, supporting fast, realistic real-time

DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2010/009-016

rendering of asteroid datasets, as well as collision detection between the asteroid and prototype surface vehicles, is an interesting challenge for computer graphics.

Because of the low and irregular gravity field of an asteroid which limits traction, hopping robots are preferred to wheeled rovers for surface exploration, since they can move themselves by making short jumps from point to point across the surface. Depending of the particular asteroid and the particular motor forces employed, a single robot hop could result in a short or a long jump, or in an orbit around the asteroid itself. Visualizing this type of behavior requires a terrain model that works over a wide range of scales as the spacecraft model interacts closely with a succession of disparate small regions of the surface. Such a terrain model is well-represented by a rapidly adaptive multiresolution data structure.

In this work we describe the integration of a suitably modified version of the Batched-Multi-triangulation framework [CGG*04, CGG*05] into a real-time simulator. The new structure embeds a fine-grained per-patch spatial index within a coarse-grained patch-based structure . The coarse-grained structure is used for fast batched I/O and rendering, while the per-patch fine-grained structure is used to accelerate raycasting and collision queries. To perform a dynamic simulation over an asteroid also requires a gravity computation, since, unlike spherically-shaped bodies, an asteroid's surface gravity varies greatly from point to point. To simulate this behavior, the gravity strength experienced by the robot is estimated at run-time according to its position with respect to the asteroid body, using a homogeneous distribution of inner points generated from the root of the multiresolution data structure. The entire system has been tested using a simple spacecraft hopper landing and exploration simulator, achieving over 130 fps in giga-triangle sized datasets of both the 25143 Itokawa and 433 Eros asteroids.

2. Related work

Given the potentially massive size of high-resolution asteroid datasets and the wide range of scales at which a simulator has to operate, it is essential for the system to be based on an adaptive level-of-detail (LOD) structure maintained out-of-core.

A general framework for managing continuous LOD models is the multi-triangulation technique [DMP98], which is based on the idea of encoding the partial order describing mutual dependencies between updates as a directed acyclic graph (DAG). In the DAG, nodes represent mesh updates (removals and/or insertions of triangles that change the representation of a mesh region), and arcs represent dependency relations among updates. Most of the continuous LOD models can be expressed in this framework, and many variations have been proposed. Until recently, however, the vast majority of view-dependent level-of-detail methods were all based on multi-resolution structures where LOD decisions are taken at the triangle/vertex primitive level. This kind of approach makes detail selection the performance bottleneck in the entire rendering process, resulting in a CPU-bound rendering pipeline to the GPU (Graphics Processing Unit). This problem is further exacerbated in rasterization approaches, because of the increasing CPU/GPU performance gap. To overcome the detail selection bottleneck, and to fully exploit the capabilities of current hardware, it is necessary to select and send batches of geometric primitives to be rendered using only a few CPU instructions. To this end, various GPU-oriented multi-resolution structures have been recently proposed. The methods are based on the idea of shifting the granularity of the representation from triangles to triangle patches [CGG*04, YSGM04]. Thus, instead of working directly at the triangle level, the models are first partitioned into blocks containing many triangles from which a multi-resolution structure is then constructed among partitions. By carefully choosing appropriate subdivision structures and managing boundary constraints, crack-free adaptive models can be constructed.

The benefit of these approaches is that the needed per-triangle workload to extract a multi-resolution model is reduced by orders of magnitude. The small patches can be preprocessed and optimized off line for a more efficient rendering, and highly efficient retained-mode graphics calls can be exploited for caching the current adaptive model in video memory. Recent work has shown that the resulting vast performance increase in CPU/GPU communication yields greatly improved frame rates [CGG*04, YSGM04, CGG*05]. The success of these coarse level approaches indicates the increasing importance of memory/bandwidth management issues in real-time rendering applications. Even though coarsening multiresolution granularity reduces the model flexibility and requires more triangles to achieve a given accuracy, the overall rendering efficiency of the system is dramatically increased rather than reduced, since rendering time does not depend linearly on triangle count anymore. Instead, rendering time is strongly influenced by how the triangles are organized in memory and sent to the graphics card.

While the coarse-grained approach improves performance for rendering, a fine grained structure is still necessary for point queries, which are used by the simulator for raycasting and collision detection. To provide this, we augment the coarse-grained structure, which organizes the rendering nodes, with a fine-grained space partitioning structure within each node, which spatially indexes each of the contained triangles. The coarse multiresolution structure is based on a diamond hierarchy, similar to the one used in Batched Adaptive Meshes [CGG*05] and constructed with an Adaptive Tetrapuzzles approach [CGG*05]. For more information on diamond multiresolution data structures, we refer the reader to a recent survey [WDF10]. The fine spatial index structure, by contrast, is based on an axis-aligned bounding box hierarchy. A similar approach, based on BSP trees, has been proposed by Lauterbach et al. [LYM07] for Interactive Ray Tracing applications.

3. Methods and tools

3.1. Multiresolution model

Management of high-fidelity representations of asteroid terrain data poses significant challenges to the design of rendering and simulation environments. In particular, since their shape is highly non-spherical, many of the simplifying assumptions used in standard planetary-scale renderers, based on digital elevation models (DEMs), do not hold. While DEMs are sufficient to represent large near-spherical terrains such as planets, they are not capable of accurately representing highly irregular surfaces such as asteroids.

In this work we propose a more general-purpose algorithm and data structure, based on the Batched-Multi-triangulation framework [CGG*04, CGG*05], allowing a high-performance model of an asteroid to be rendered at interactive rates down do a spatial resolution of a few centimeters. Extending this same model is also the basis for a collision-detection algorithm used to drive the physics-based simulation of a small-body surface robotic spacecraft. The underlying idea of the method is to depart from current point or triangle-based multiresolution models and adopt a patch-based data structure, from which view-dependent conforming mesh representations can be efficiently extracted by combining precomputed patches. Since each patch is itself a mesh composed of a few thousand triangles, the multiresolution extraction cost is amortized over many graphics primitives, and CPU/GPU communication can be optimized to fully exploit the complex memory hierarchy of modern graphics platforms. In order to also accelerate raycasting and collision queries, we augment this coarse grained structure with a per-patch spatial index that organizes individual triangles in the patch triangle strip.

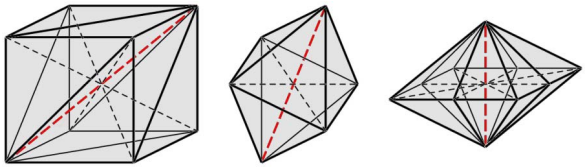

Figure 2: ***Diamond subdivision structure.*** *Diamonds represent tetrahedrons that must be atomically split.*

For construction, we follow the Adaptive Tetrapuzzles approach [CGG*04]. The method uses a conforming hierarchy of tetrahedrons generated by recursive longest edge bisection to spatially partition the model, where each tetrahedral cell contains a precomputed simplified version of the original model. The representation is constructed off-line during a fine-to-coarse parallel out-of-core simplification of the surface contained in diamonds (sets of tetrahedral cells sharing their longest edge, see Fig. 2). Appropriate boundary constraints are introduced in the simplification process to ensure that all conforming selective subdivisions of the tetrahedron hierarchy lead to correctly matching surface patches. At run-time, selective refinement queries based on projected or world space error estimation and regions of interest are performed on the external memory tetrahedron hierarchy to rapidly produce view-dependent continuous mesh representations by combining precomputed patches. These queries are implemented using a dual-queue algorithm applied to a diamond graph (see Fig. 3 for the graph structure).

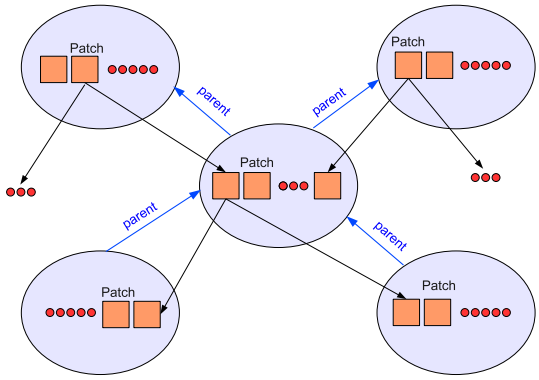

Figure 3: ***Diamond graph structure.*** *Each node represents a group of patches that must be atomically split. Dependencies among nodes are encoded in a DAG.*

The resulting technique has the following properties: it is fully adaptive and is able to retain all the original topological and geometrical detail, even for massive datasets; it is not limited to meshes of a particular topological genus or with a particular subdivision connectivity and it preserves geometric continuity of variable resolution representations at no run-time cost; it is strongly GPU-bound and is over one order of magnitude faster than existing adaptive tessellation solutions on current PC platforms, since its patch-based structure successfully exploits on-board caching, cache coherent stripification, compressed out of core representation and speculative prefetching for efficient rendering on commodity graphics platforms with limited main memory; it enables high quality simplified representations to be constructed with a distributed out of core simplification algorithm. At run-time, adaptive cuts of the diamond graph lead to variable resolution surface models, which can be exploited both for rendering and for collision/ray intersection queries.

The patch-size granularity of the method is efficient enough to ensure interactive and high quality rendering but too coarse for collision detection and dynamics simulation. For this reason a fine grained BSP structure is maintained within each node in order to spatially index individual triangles, helping and speeding up the raycasting and reach a triangle-size granularity (see Fig. 4). This BSP structure is constructed on-the-fly at patch loading time using a fast recursive split procedure. Given N triangles in a patch, these are organized in a single generalized cache coherent triangle strip of $M \geq N+2$ vertices. We recursively split each

strip at the median edge in order to define a balanced tree on the strip. At each step, we record the left and right bounding boxes. This defines a balanced spatial bounding box tree on the patch mesh, such that only the two bounding boxes must be stored. At run-time, the tree can be used for ray-casting or collision queries implemented with top-down descents.

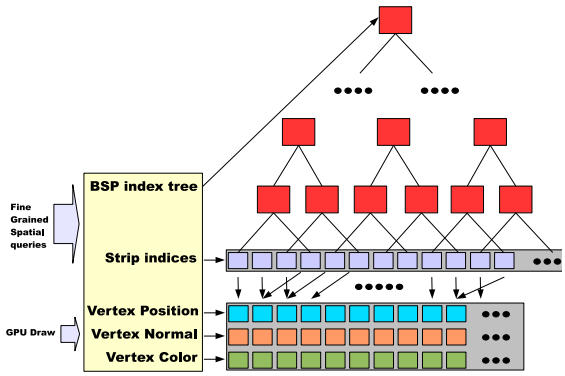

Figure 4: ***Patch structure.*** *Each patch is coarse grained, but also contains a fine grained spatial index.*

Implementation of the algorithm to work with third-party rendering engines is accomplished using a visitor software pattern, which permits the application of callbacks to each of the nodes in the current graph cut.

3.2. Application integration

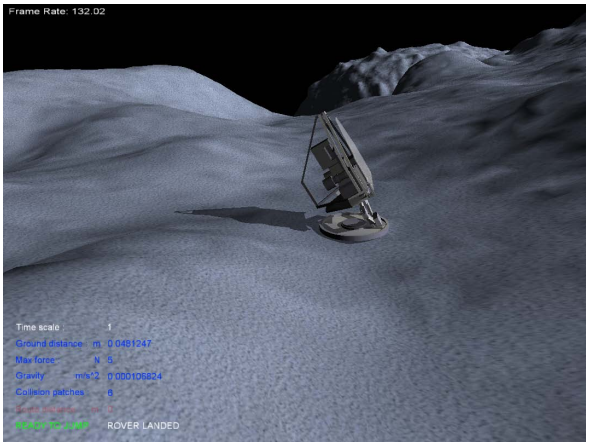

Figure 5: ***Frogbot ready to jump over Itokawa surface.*** *Screenshot taken during a simulation using the asteroid Itokawa 5cm res dataset (about 220 Mtriangles).*

The application architecture is divided in two standalone executable components: an out-of-core parallel database construction tool and real-time data viewing and simulation tool. The out-of-core component constructs a multiresolution database starting from a high resolution mesh. The first phase of this building is a mesh partition using a hierarchical space partitioning scheme, generating a binary forest of tetrahedrons. The forest is built in a top-down fashion, through recursive insertion of mesh triangles, starting from an initial subdivision of the mesh bounding box into six tetrahedrons around a major box diagonal. The second and final phase is carried out by constructing non-leaf cells through bottom-up recombination and simplification of lower level cells and assigning model space errors and bounding volumes to them.

Once the multi-resolution database has been constructed by the out-of-core component, the viewing and simulating tool can perform real-time multiresolution queries against it, using a variety of different strategies and data visitors. Since our target simulation software environment uses the open source Delta3D engine, the simulation application includes several customized data access methods for generating runtime Delta3D objects from the multiresolution dataset.

In general, there are two alternative ways for the simulation application to access the multiresolution database, the first one being for the view-dependent rendering and the second one being for the collision detection. This behavior has been implemented in each case by defining a custom visitor and a custom refinement strategy in order to generate the appropriate Delta3D objects. The first of these data access visitors used by the simulator is responsible for the surface visualization. By using a view-dependent/pixel tolerance refinement strategy, with a typical target accuracy of one triangle per pixel, it updates the Delta3D scene graph with the terrain patches at the currently appropriate LOD. The second data access visitor is used for detecting collisions, and is driven by the position and velocity of the surface robot as it interacts with the terrain. This visitor updates the Delta3D collidible objects by using a world space refinement strategy. This strategy selects triangles with a centimetric edge length within a bounding box centered at the current simulated object position. The selection of the current graph cut is realized using a dual queue algorithm [DWS*97] applied to the out-of-core graph. The newly requested portions of the graph are asynchronously loaded by a separate thread. Pre-fetching is applied to avoid stalling the renderer/simulator.

Because of the extremely low and highly irregular gravity field of a small body such as an asteroid (see Fig. 6), it is very difficult to move over its surface using a wheeled rover. Therefore, in order to create a simulation application, we have constructed a simple notional robotic hopper model based on a published Jet Propulsion Laboratory prototype concept [HSBF00]. The model, which we call the *frogbot*, consists of eleven movable parts: 1 head, 3 flaps, 2 legs (2 parts for each leg), 1 bearing, 1 leg base and 1 foot. The frogbot has the capability to store energy in a geared 6-bar spring/linkage system and to jump like a frog by releasing its legs (see Fig. 11). Additional motors drive the foot bearing and flaps for self-righting the robot if it lands on its side.

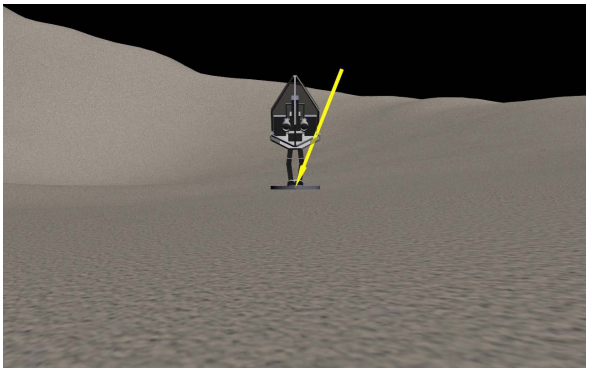

Figure 6: ***Eros gravity vector direction.*** *The image shows the gravity vector direction on Eros's surface. The estimated gravitational force at this point during the simulation is* $0.0061\frac{m}{s^2}$.

Depending on the particular small body surface gravity and the motor forces employed, the jump could result in a short jump to a nearby location, a long jump to a distant location, or departure from the small body completely by achieving escape velocity. Rendering the surface through such a variety of possible motions necessitates a rapidly adaptive multiresolution structure, as is provided by the multiresolution database. On the other hand, simulation behavior calculations such as landing and interacting with the surface of an asteroid require collision detection or raycasting operations at a finer granularity level than that provided by patch-size data structures. Usually this task requires high computational resources, especially for large geometries. To reduce the granular size of the structure, an independent BSP structure of triangles is generated for each patch run-time during the graph cut extraction in order to speed up the raycasting and reach a triangle-size granularity.

Another important calculation necessary to simulate orbiting, as well surface reaction forces, is gravity computation. Unlike spherically-shaped bodies, whose gravitational force can be accurately modeled as a uniform central force, a small body's gravitational field is highly irregular, and its surface gravity can vary considerably from point to point. The gravitational force experienced by the robot is estimated at run-time according to its position with respect to the asteroid body. This is done by discretizing the total asteroid mass m_a into N discrete points inside the volume, calculating Newton's law of gravity for each point and summing up the results to determine the net gravitational force:

$$\mathbf{F}(\mathbf{P}) = \sum_{i=0}^{N} G\frac{m_a}{N}\frac{(\mathbf{P}-\mathbf{P_i})}{\|\mathbf{P}-\mathbf{P_i}\|} \quad (1)$$

where G is the universal gravitational constant and the mass particles P_i are generated by uniformly distributing random points in the inside of the body (see Fig. 7). Each point

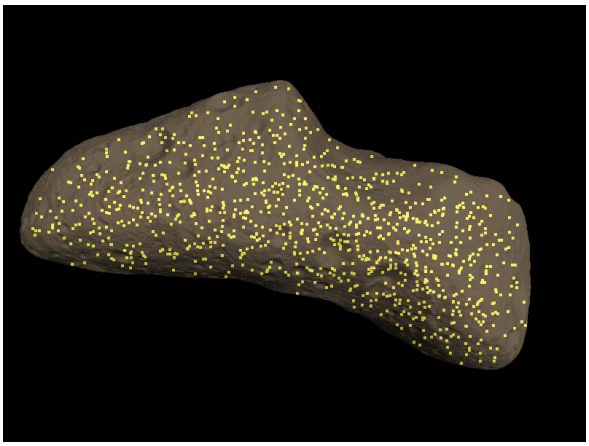

Figure 7: ***Mass particles.*** *The gravitational force experienced by the robot is estimated at run-time according to its position with respect to the asteroid body. This is done by distributing the total mass of the asteroid among uniformly distribute particles within its volume and summing the individual force contributions.*

contribution is a function of the robot position and is computed on the fly. The sum can be accelerated using a hierarchical method [BH86] by constructing a BSP tree on top of the mass particle list. Mass particles are generated by taking uniformly distributed random points within the bounding box of the asteroid, and testing them for inclusion within the asteroid body using a fast point in polyhedron test. Pseudo-random points with low discrepancy are generated using a Halton sequence [KN05], while the point-in-polyhedron test uses a ray-casting algorithm, which counts, how many triangles are intersected by a ray starting from the tested point. Since the ray-casting process uses the acceleration structures embedded in our graph, point generation is fast.

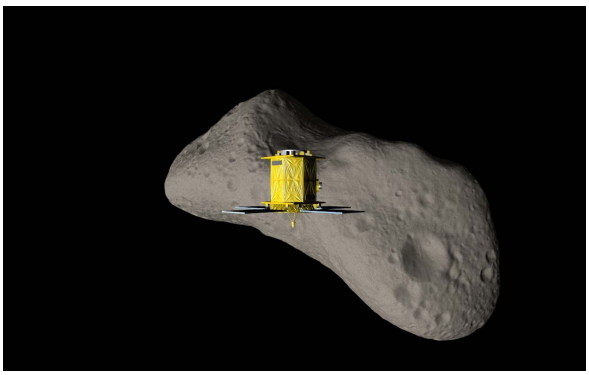

Figure 8: ***Simulation example: Animation of the NEAR probe approaching Eros.*** *Image is a screen shot taken from a real-time simulation.*

The simulation application supports two simulation

modes: in the first mode, the user can explore the entire asteroid surface by flying and orbiting the spacecraft around it (see Fig. 8) at high interactive frame rates. In the second mode, after having chosen a place to land, the robot's surface exploration can be controlled by interactively commanding it to jump from site to site on the surface. Numerical result and frame rates for both the orbit and ground exploration modes are detailed in the following section.

4. Results

A terrain database builder and a software library supporting an OpenGL viewer and a Delta3D application have been implemented using C++, on both Linux and Windows platforms. A simple Delta3D simulation application demonstrating the library's capabilities has also been created, the capabilities of which include interactive navigation around the asteroid and dynamics simulation of the robot moving on the ground.

We have tested the system with the two asteroid shape models present in the freely available NASA planetary Data System (http://sbn.psi.edu/pds/archive/neos.html).
The shape model of 25143 Itokawa is derived by Robert Gaskell from Hayabusa AMICA (Asteroid Multi-band Imaging CAmera) images [Gas08] and the shape model of 433 Eros is derived by Robert Gaskell too from NEAR MSI images [GSI*08]. The original models are provided in the implicitly connected quadrilateral (ICQ) format with four levels of resolution and converted in a triangulated version of 3145728 triangles. For hopper simulations, higher resolution versions have been created by a supersampling process with introduced procedural detail: Eros is supersampled at 1.5 meter/sample (ratio 470) with 1.5 billion triangles (see Fig. 10) and Itokawa is supersampled at 5cm/sample (ratio 70) with 220 million triangles (see Fig. 9).

4.1. Preprocessing

	Triangles	Patch Size	CPU	Tetrahedra	Time	Output Size
Itokawa 5cm	220M	8000	i7 sequential	438162	6h 35m	6.8 GB
Itokawa 5cm	220M	8000	cluster mpi 1+1	438162	8h 20m	6.8 GB
Itokawa 5cm	220M	8000	cluster mpi 1+4	438162	4h 52m	6.8 GB
Itokawa 5cm	220M	8000	cluster mpi 1+8	438162	2h 31m	6.8 GB
Itokawa 5cm	220M	8000	cluster mpi 1+28	438162	2h 16m	6.8 GB
Eros 1.5m	1.5G	16000	cluster mpi 1+28	1152210	11h 18m	21.1 GB

Table 1: ***Numerical results for asteroid preprocessing.*** *Statistics acquired on a sequential build on an Intel i7 960 machine with 24GB memory and several parallel builds on a PC cluster with 1 master and 28 nodes, where each machine has an Intel Core2 Quad Q9550 2.83 GHz with 4GB memory. The Eros dataset presented is larger than the Itokawa dataset (1.5G triangles vs. 220M) and the build has been performed for practical reasons only on the cluster. The Itokawa data structure takes 6.8GB of disk space with a tetrahedron/patch size of 8000 triangles, while Eros dataset takes 21.1GB with patches of 16000 triangles.*

Table 1 lists numerical results for several out-of-core preprocessing runs on the datasets. The tests were executed on an Intel i7 960 3.20 GHz multicore based machine with 24GB memory and on a PC cluster with 1 master and 28 nodes, where each machine has an Intel Core2 Quad Q9550 2.83 GHz with 4GB memory. The Itokawa dataset was built using 8000 triangles for each patch, whereas the Eros dataset was built with 16000 triangles per patch. Table 1 shows the scalability of the Itokawa build increasing the workers count (1,4,8,28), which confirms the efficiency of the distributed approach. The speed-up is linear until about 8 processors, and decreases later when disk I/O becomes dominant.

4.2. Rendering and Simulation

	Mode	FPS	Tri/sec	Rendering	Collision	Rendered
				patches	Patches	Triangles
Itokawa 5cm	only rendering	331	56.3M	24	0	169.7k
Itokawa 5cm	only rendering	110	190.5M	378	0	1.7M
Itokawa 5cm	only rendering	128	180.0M	318	0	1.4M
Itokawa 5cm	rendering and dynamics	140	113.5M	200	8	2.0M
Eros 1.5m	only rendering	338	29.0M	6	0	85.8k
Eros 1.5m	only rendering	93	192.1M	190	0	2.0M
Eros 1.5m	only rendering	169	185.4M	100	0	1.1M
Eros 1.5m	rendering and dynamics	130	166.5M	90	7	2.1M

Table 2: ***Numerical results for Rendering and Simulation.*** *Statistics acquired along a path from orbit to ground contact, using a camera placed on the top of the probe for the far orbit view and a panoramic third person camera after the landing.*

Table 2 lists some rendering numerical results of the demo simulator application. We evaluated the rendering and dynamics performance of the system using different versions of the dataset, varying the model resolution and the triangle patch size. The simulator has been tested on a PC with an Intel Core2 Quad Q6600 2.40GHz machine with 4GB memory and NVIDIA GeForce GTX 280, both with Linux and Windows. Increasing the model resolution (from 20cm to 5cm for Itokawa and from 5m to 1.5m for Eros) caused no change in simulator performance, with an average rendering performance of 135Hz. The application speed is clearly driven by the dynamics engine performance (Delta3D using ODE - Open Dynamics Engine), depending on the collisions detection state and the collision meshes involved.

The dynamics frogbot model representation consists of a small collection of simple part shapes (e.g. boxes, spheres, cylinders) in order to have contact only between simple shapes and generic triangle meshes and therefore avoid expensive mesh to mesh contacts. Additionally, we tried out different triangle patch sizes (from 24000 to 8000 triangles/patch), and have experimentally determined that the ODE dynamics engine works better with patches with less than 16000 triangles. In both the Eros and Itokawa cases, the frame rate has a minimum point of 60fps during the beginning of a full contact and an average frame rate of 135fps when the robot is stationary on the ground or flying. All the tests have been performed using a screen window size of 1280x800 pixels, hardware full scene antialiasing (8x - 4xMS,4xCS) and a screen tolerance of 2 pixels as rendering strategy settings.

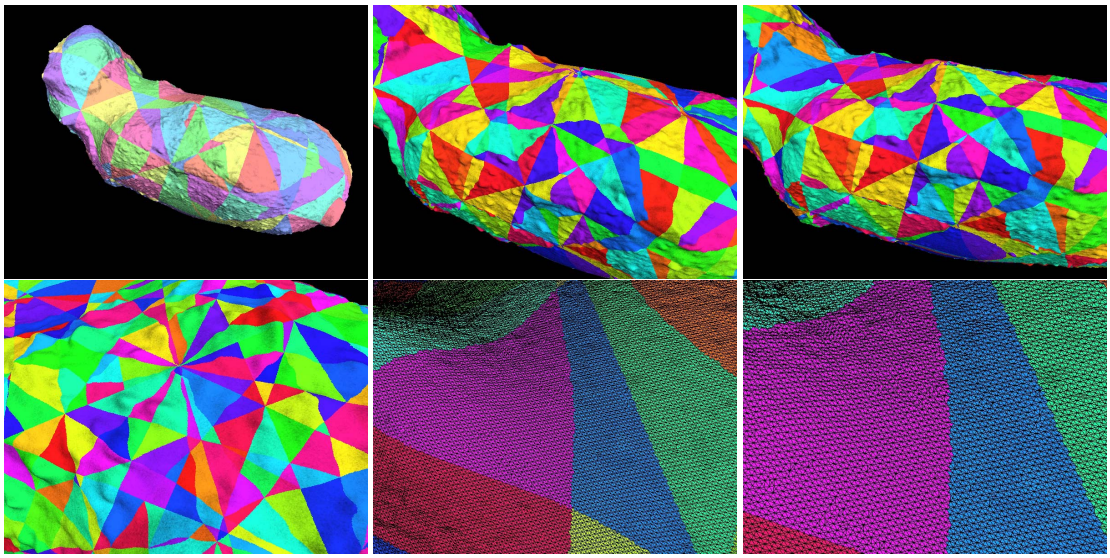

Figure 9: ***Real-time rendering of Itokawa.*** *Selected screenshots at various scales obtained during real-time navigation. The total dataset size is 220M triangles. A coarse tolerance is used to make tessellation visible.*

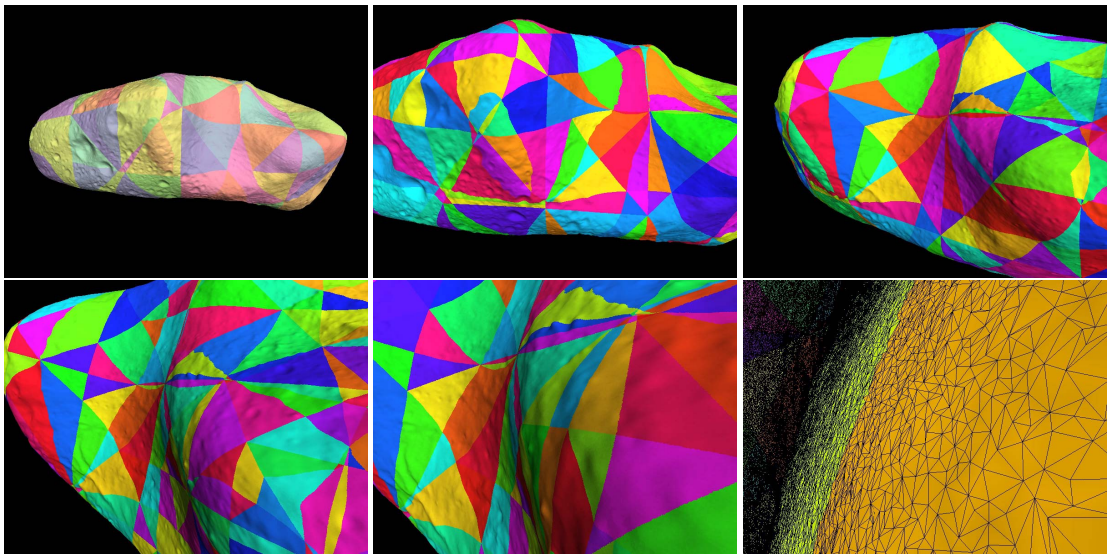

Figure 10: ***Real-time rendering of Eros.*** *Selected snapshots at various scales obtained during real-time navigation. The total dataset size is 1.5G triangles. A coarse tolerance is used to make tessellation visible.*

For the dynamics simulation part, the world space refinement strategy uses a tolerance in centimeters, smaller than the model resolution.

5. Conclusions and Future Work

We have presented an enhancement of the Batched Multi-Triangulation technology in order to integrate it into a high-performance small-body surface robotics simulation software environment. The resulting simulation software environment and can allow a user to visualize and control a simulated robot in real-time as it traverses an extremely high-resolution asteroid surface. Several features of the enhanced batched multi-triangulation technology make it well-suited to the unusual requirements of such a simulation. The batched multiresolution structure provides interactive and high quality rendering of a terrain model with a resolution on the order of a centimeter as the robot hops randomly about, or smoothly orbits the surface. At the same time the, internal patch BSP structure extends the granularity of the batched system to support the point-sampling necessary for the dynamics simulation.

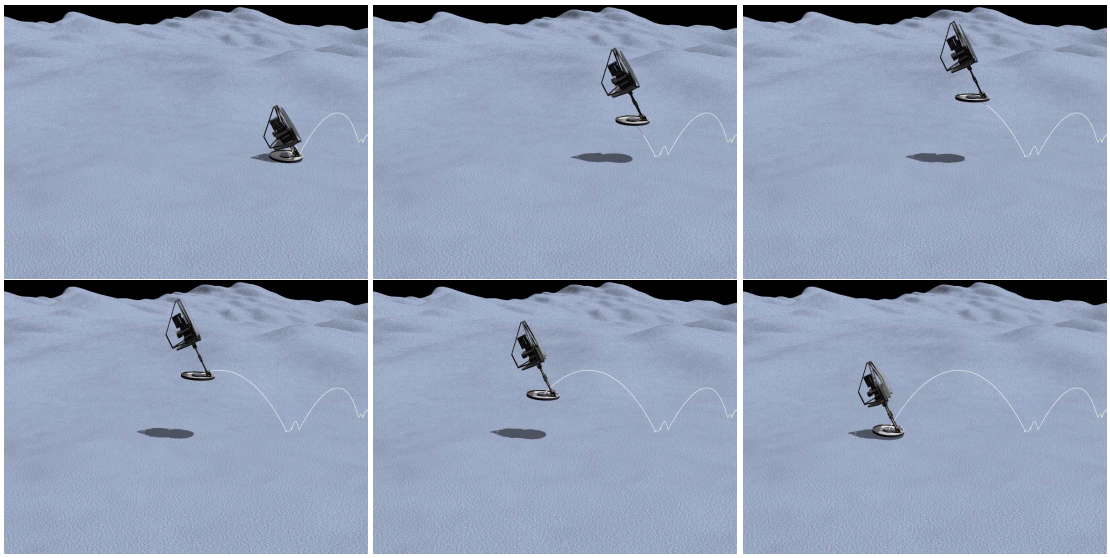

Figure 11: ***Demo frogbot jumping over Itokawa.*** *Image sequence taken from a live simulation showing the frogbot charging legs, releasing them and then flying.*

For future work, we would like to create a more realistic simulation scenario by using a more sophisticated terrain supersampling algorithm and a more physically accurate robot model based on a working mechanical prototype. We would also like to improve the fidelity of the collision response algorithm, ideally using physically accurate material properties based on empirical data. As a result of the high performance rendering and collision detection delivered by the multi-resolution technology, the simulation performance bottleneck has become dominated by the dynamics engine performance. Therefore, we would also like to explore future use of a more sophisticated dynamics engine to improve the performance and realism of the robot-surface interaction. One such engine we are studying is the Bullet open source physics engine (http://bulletphysics.org).

Acknowledgments. This work was partially sponsored by the International Supplementary IR&D program of the Johns Hopkins University Applied Physics Laboratory under contract number 967735, in support of research led by Stuart Hill of the JHU-APL Space Department and sponsored by its Civilian Space Business Area.

References

[BH86] BARNES J., HUT. P.: A hierarchical O(N log N) force-calculation algorithm. *Nature 324*, 4 (dec 1986).

[CGG*04] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 796–803.

[CGG*05] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Batched multi triangulation. In *Proceedings IEEE Visualization* (Conference held in Minneapolis, MI, USA, October 2005), IEEE Computer Society Press, pp. 207–214.

[DMP98] DE FLORIANI L., MAGILLO P., PUPPO E.: Efficient Implementation of Multi-Triangulations. In *Proc. IEEE Visualization* (1998), pp. 43–50.

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: ROAMing terrain: real-time optimally adapting meshes. In *Proc. IEEE Visualization* (1997), pp. 81–88.

[Gas08] GASKELL R.: *Gaskell Eros Shape Model V1.0. NEAR-A-MSI-5-EROSSHAPE-V1.0*. Tech. rep., NASA Planetary Data System, February 2008.

[GSI*08] GASKELL R., SAITO J., ISHIGURO M., KUBOTA T., HASHIMOTO T., HIRATA N., ABE S., BARNOUIN-JHA O., SCHEERES D.: *Gaskell Itokawa Shape Model V1.0. HAY-A-AMICA-5-ITOKAWASHAPE-V1.0*. Tech. rep., NASA Planetary Data System, February 2008.

[HSBF00] HALE E., SCHARA N., BURDICK J., FIORINI P.: A minimally actuated hopping rover for exploration of celestial bodies. In *Proc. IEEE International Conference On Robotics and Automation* (2000), pp. 420–427.

[KN05] KUIPERS L., NIEDERREITER H.: *Uniform distribution of sequences*. Dover Publications, 2005.

[LYM07] LAUTERBACH C., YOON S.-E., MANOCHA D.: Ray-strips: A compact mesh representation for interactive ray tracing. In *IEEE/EG Symposium on Interactive Ray Tracing* (2007), pp. 19–26.

[WDF10] WEISS K., DE FLORIANI L.: Simplex and diamond hierarchies: Models and applications. In *Eurographics State of the Art Reports* (2010), Eurographics Association, pp. 113–136.

[YSGM04] YOON S.-E., SALOMON B., GAYLE R., MANOCHA D.: Quick-VDR: Interactive View-Dependent Rendering of Massive Models. In *Proceedings of IEEE Visualization 2004* (2004), pp. 131–138.