2018 15th International Conference on
Control, Automation, Robotics and Vision (ICARCV)
Singapore, November 18-21, 2018

AstroGen – Procedural Generation of Highly Detailed Asteroid Models

Xi-zhi Li, René Weller, Gabriel Zachmann

Abstract— **We present a novel algorithm, called AstroGen, to procedurally generate highly detailed and realistic 3D meshes of small celestial bodies automatically. AstroGen gains it's realism from learning surface details from real world asteroid data. We use a sphere packing-based metaball approach to represent the rough shape and a set of noise functions for the surface details. The main idea is to apply an optimization algorithm to adopt these representations to available highly detailed asteroid models with respect to a similarity measure. Our results show that our approach is able to generate a wide variety of different celestial bodies with very complex surface structures like caves and craters.**

I. INTRODUCTION

The study of small celestial bodies in the solar system (e.g. asteroids, comets) has become an area of great interest for astronomy science in the past decades. For instance, Galileo and Cassini are successful missions to investigate small celestial bodies and collected abundant interesting data about several asteroids. More recently, JAXA's Hayabusa2 spacecraft will, after studying Ryugu in depth from orbit for about a year, drop three rovers and a lander onto the asteroid's surface this month and hopefully, some samples will be sent back.

Such long distance missions are challenging for several reasons: first, the communication takes a very long time. Hence, it is not possible to immediately react on complications during e.g. the landing phase by the mission control on earth. Consequently, spacecrafts operating in such environments are usually equipped with some sort of autonomy. Second, the terrain of the asteroids is usually unknown in advance during the planning phase of the mission. The most economic and common way to observe asteroids from earth is to obtain data radar or lightcurve inversion. However, these methods do not deliver surface details. Nevertheless, the landing spacecrafts and rovers has to be designed with such very limited information.

Usually, space missions are planned with support of virtual testbeds (VTBs) before building physical mock-ups (See Fig. 1). These testbeds consist of physically-based simulation of terrain that provide a real-time, immersive and 3D interaction environments which give engineers the opportunity to interact with the simulated spacecraft or rover to gain comprehensive understanding of possible design flaws during the early design process as well as later mission stages like training and supervision [1]. In order to simulate a large amount of possible scenarios to be prepared for a lot of circumstances, it is essential to have a large number of highly detailed and realistic 3D asteroid models available in such a virtual testbed. In the case that autonomous algorithms on board of the spacecraft support the landing operation or the energy-efficient navigation of a rover on the terrain, there is even more need for such 3D models because these AI algorithms usually have to be trained with a large amount of such data.

The generation of such models has two major challenges: the lack of ground truth data and the missing of appropriate methods to synthesize realistic models of irregular shaped small bodies. Actually, the only available highly detailed asteroid model is that of Itokawa[1]. Moreover, traditional terrain generation algorithms are almost all optimized for spheroidal planetary models where a simple heightmap can be used. This is not directly applicable to irregular celestial objects such as asteroids especially if terrain details such as caves have to be considered.

We present, to our knowledge, the first algorithm that is able to compute realistic highly detailed 3D models of irregular celestial bodies fully automatically. The main idea is to combine different implicit object representations with an optimization algorithm to adapt their implicit parameters to real world models. These parameters can be varied or applied to completely different basic shapes while remaining the overall surface texture.

More precisely, we use a two-tier approach: For a given ground truth asteroid shape, we first approximate the basic asteroid shape by a polydisperse sphere packing. A first optimization step adapts parameters of a metaball approach. In a second optimization step, we learn the surface details by optimizing the parameters of several noise functions that are well chosen to represent typical surface structures of celestial bodies such as craters and caves. This makes it easy to transfer the surface details to other basic shapes. During the training phase, we allow small intervals of all parameters with respect to the distance function. This enables us to further vary the surface details but also the underlying metaball shape.

Our algorithm, called *AstroGen* supports:

- *Full automatic generation* of almost endless variations for a given ground truth asteroid model within a predefined error bound. However, the parameters of the

This work was not supported by any organization
X.Z. Li, University of Bremen, Germany lixizhi@uni-bremen.de
R. Weller, University of Bremen, Germany weller@cs.uni-bremen.de
G. Zachmann, University of Bremen, Germany zach@cs.uni-bremen.de

[1]The data from the Rosetta mission was published simultaneously to this submission. Hence, we could not include this in our algorithm. However, our algorithms would obviously also work with this model.

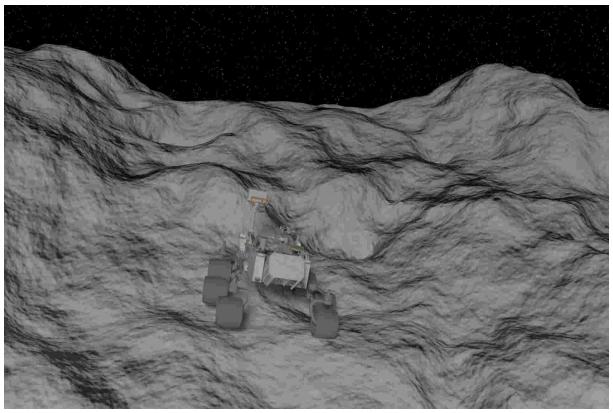

Fig. 1. Simulated rover cruise on the virtual testbed of unknown asteroid stein [2].

synthetization algorithm are easy to understand which makes it easy to manually adjust the generated models.
- *High performance* due to an almost full GPU implementation of all time consuming parts of the algorithm.
- *Arbitrary Resolutions:* due to the implicit representation it is easy to generate polygonal models at any resolution with the marching cube algorithm.

In a use case scenario we have applied AstroGen to the currently only available high resolution asteroid model of Itokawa and generated several variations of it. Moreover, we applied the surface details to low-poly models of Stein [2], Ceres [3] and Lutetia [4].

II. RELATED WORK

There are four main approaches to generate terrain features: procedural, physics-based, sketch-based and example-based [5]. Physics-based methods generate terrain features geologically correct, but often computationally expensive and lack of scalability in simulation different natural phenomenons. Sketch-based methods require heavy manual intervention while example-based methods are limited by the input data and restricted to two-dimensional terrain features [5]. In contrast, procedural methods are fast and easy to generate arbitrary resolution of realistic terrain in the virtual world, see [6], [7] for a broader overview. Often, such methods rely on some kind of noise function. Perlin noise is known as efficient and its inherent self-similarity, consistency properties are suitable for terrain generation [8]. Enhanced version, like simplex noise [9] reduce some artifacts or generate particular terrain features, such as ridges or rolling hills [10]. The Commercial software Terragen [11] and e.g. a bunch of researcher papers [12], [5], [13] created diverse and realistic terrain based on noise method. Togelius [12] introduced evolutionary algorithm with noise method to generate terrain map and balanced on several objectives, such as playability and realistic of terrain. [5] focused on Hydrology terrain simulation by using fractal interpolation to connect predefined physical-based terrain features such as river networks, mountain ridges and valleys. [13] proposed a method based on real elevation statistics and utilize value noise – a variant of perlin noise – to generate geotypical terrain. Recently, compact mathematical definition [14] and sparse procedural [15] method are proposed which efficiently combine different terrain primitives and give user more intuitive control about the scene. However, noise-based methods usually tend to create terrain that is uniform at fixed amplitude and frequency values, and often require massive post-process to generate interesting features and choosing the correct parameters for this post-processing is often unintuitive. Moreover, none of these algorithms supports the generation (or variation) of irregular celestial bodies.

III. OUR APPROACH

The goal of our algorithm is to generate a wide variety of different asteroid models considering the underlying basic irregular shape as well as the surface details. We want to achieve high realism and detailed high polygon models. In order to guarantee realism, AstroGen relies on the usage of real world data, namely surface details from previous space missions, like that of Itokawa and data from earth observation that delivers the rough shape and can be found in extensive asteroid databases [16]. Due to the flexibility of our algorithm, it is easy to also include more data (like that from the Rosetta mission that was just released or from the Hayabusa2 mission).

The two different kinds of data already suggest a two-tier approach. Consequently, AstroGen consists mainly of two stages:

1) We use an *implicit shape* approach to represent the underlying rough shape. The advantage is that it easily allows to make small variations in the rough shape and additionally, we can generate high-poly meshes from it.
2) The surface details are represented by different *noise functions*. Again, this enables us to generate poly meshes in arbitrary resolution.

In order to adapt our asteroids to the real world data mentioned above, we use an *optimization algorithm* to optimize the parameters of our rough shape as well as the surface details. However, we allow small variations in the parameter range to generate an almost infinite number of variations. Finally, we present a method to generate a polygonal mesh from our implicit asteroid representation based on marching cubes.

All these steps can be performed entirely massively parallel on the GPU what guarantees a high performance. However, the optimization steps are required only once per ground truth data. For the generation of a wide variety of asteroid models it is sufficient to simply vary the parameters and generate a mesh using the final step in our algorithm. Fig. 2 shows an overview on the complete process. In the following, we will present the details of the individual steps of our automatic pipeline.

A. Implicit Shape Representation

Generally, there are two main types of method to represent a free-form surface: *parametric surfaces* and *implicit sur-*

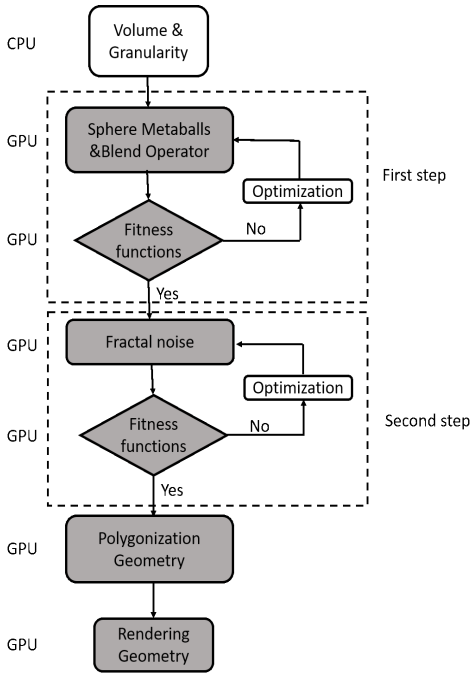

Fig. 2. The pipeline of two parts simulation.The dark box means the program running on the GPU while the white box on the CPU side.

faces. In the first case, Hermite-spline, B-spline and NURBS are most commonly used. For the latter, metaballs, skeleton and convolution surfaces are the most popular methods. Implicit surface modeling produce smooth and aesthetic shapes that can be seamlessly modified while keeping a consistent structure. In addition, their function definition is compact and require quite simple primitives such as sphere or ellipsoid to construct a model which is suitable for real-time simulation. Finally, it is possible to adapt this representation to existing polygonal shapes, like the low-poly models from the asteroid database. These are the main reasons, we decided to use an implicit surface representation for the basic shape of our celestial bodies.

In principal, implicit surfaces define a $\Re^2$ 2-D manifold, a surface S embedded in the three-dimensional space $\Re^3$:

$$S = \{(x,y,z)|F(x,y,z) = T\} \tag{1}$$

For a *skeleton implicit surface*, we usually have given a set of distinct n constraint points $c_1,...,c_n$, and a set of potential functions F(c_i) for each point. These define a smooth surface M with:

$$M = \{(c)|F(c_i) = \sum_i^n \omega_i F(c_i)) + t\} \tag{2}$$

The challenge is to select an appropriate set of points and potential function.

Polydisperse Sphere packing

Our idea is based on the approach by Wyvill and Brian [17] that extends the metaballs algorithm [18] by using a so-called BlobTree to represent the skeleton. However, with this method, the points are located on the medial axis of the object. This makes it complicated to add e.g. additional hills or to remove parts to form valleys. Hence, we decided to modify this idea by adopting a sphere packing-based approach. Originally invented for collision detection, the Protosphere [19] algorithm delivers a polydisperse sphere packing of arbitrary 3D objects. This is ideal for metaball-based modeling systems and for our application in particular because the greedy choice of the algorithm automatically leads to a level-of-detail representation for the model, i.e. it offers an easy trade between speed and accuracy. Moreover, it is easy to simply add or remove individual spheres to create a small variety of the basic shape. Finally, the algorithm is fast and works completely on the GPU. These spheres define our constraint points in Equation 2. Fig. 3 shows a sphere packing of Itokawa generated by Protosphere.

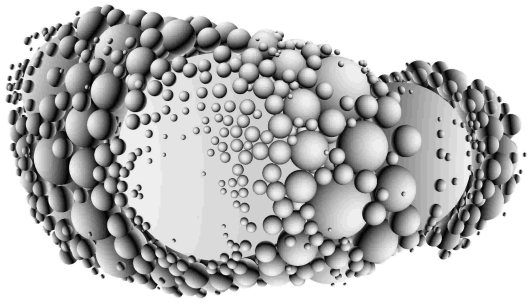

Fig. 3. The sphere packing representation of asteroid Itokawa by Protosphere [19].

Blending

After defining the constraint points, we have to define the potential functions in Equation 2. Remember, that the surface details are added in the second step, hence, we first want to create a smooth surface. To do that, we slightly modified Blinn's [18] original potential function to fit our constraint points:

$$f(r_i) = \begin{cases} e^{a-br_i^2} & if\ r_i \in [0,5R] \\ 0, & otherwise \end{cases} \tag{3}$$

with

- a and b are the tension factors that control the smoothness in the overlapping areas and the softness of each metaball
- and r_i is the radius of each spheres and $R = max\{r_i\}$

Summarizing, we get the complete potential function of P in 3D space as:

$$f(r) = \sum_{i=1}^{n} f(r_i(P,C_i)) \tag{4}$$

While implicit surface deformation is represented by multiple metaballs, bulge, crease and tearing frequently appear in overlapping areas. In order to eliminate multiple metaball influences on an overlapping area we additionally added some blending function according to [17]:

$$f(A \cdot B) = (f^m(A) + f^m(B))^{\frac{1}{m}} \tag{5}$$

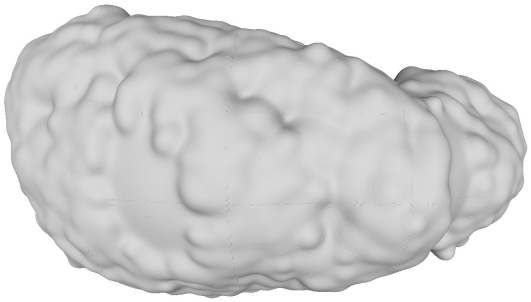

Fig. 4. The implicit metaball shape of Itokawa (892k vertices).

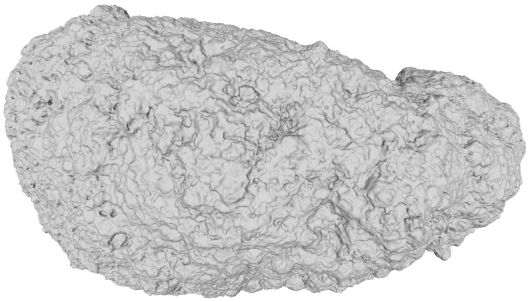

Fig. 5. The rough shape of the asteroid Itokawa with surface details generated by Perlin noise.

with

- $f(A)$ and $f(B)$ represent two metaball's potential functions
- and m controls the influence of the overlap in the distance field; With $m = 1$ we get the traditional overlapping method. In case of high convexity we can modestly change this parameter.

Fig. 4 shows the metaball surface of Itokawa.

B. Noise Based Surface Features

In order to generate a realistic surface features, we apply several noise functions to the underlying metaball model described above. In the following, we will describe the different layers of noise we used.

Perlin Noise

Perlin noise is a type of noise that is often used to generate terrain because it fulfills Tobler's First Law of Geography [20]. Basically, Perlin noise is a lattice-based gradient noise (see Equation 6). However, simple Perlin noise often leads to repetitive patterns. Hence, we use different combination of Perlin noise to created more complex terrains. The complexity of the generated terrain can be controlled by several parameters. The most important is the number of *octaves*. For a given frequency and amplitude we can generate an octave by doubling the frequency and halving the amplitude or vice versa. For instance, progressively adding lower frequencies (with higher amplitudes) generates larger terrain structures, such as large mountains and trenches [21]. Accumulation of eight octaves is called Fractional Brownian Motion (FBM). In our implementation, we used FBM. Moreover, we use *simplex noise*, a derivative of Perlin noise that uses a simplex instead of a quadrangular lattice. This improves the performance significantly. Additionally, the combination allows us to enhance the control of the generated details. Fig. 5 shows the surface created by different combinations of Perlin noise.

$$perlin(x) = \sum_{i=0}^{\infty} p^i n_i(2^i x),\ x \in [0,1],\ p \in [0,1] \qquad (6)$$

with

- p defines the scaling factor of the amplitude on successive octaves,
- 2^i controls the scaling along the x for the octaves

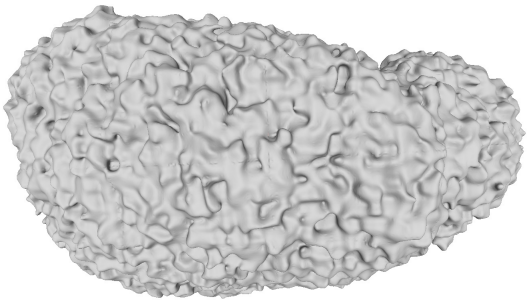

Fig. 6. The cave occurs through warping the coordinate.

- and the function $n_i()$ is a function that generates random values

Caves

Using too many octaves of Perlin and simplex noise results in isotropic details that can give the terrain an artificial look. A usual way to overcome this problem is to modulate the original shape (in our application, the rough shape generated by the metaballs) with another noise function. This technique, called *warping* is very common in computer graphics for generating procedural textures or geometry. Using medium frequencies and mild amplitudes results in surreal ropey organic-looking terrains. Lower frequencies and higher amplitudes increase the occurrence of caves, tunnels, and arches [21]. Fig. 6 shows the Itokawa model with warped coordinates.

Craters

Craters are one of the most prominent visual terrain elements of celestial bodies without atmosphere. Unfortunately, Perlin noise is not able to generate structures that look like craters, or at least, it is not known how to set the parameters until now. Consequently, we use another type of noise to support this sort of terrain.

In contrast to Perlin noise, Worley noise [22] is not gradient-based, but value-based. The basic idea of Worley noise is to grow points until another growing point is hit [23]. This leads exactly to terrains that look like craters. We adopted a simplified GPU-based version similar to [24]. Fig. 7 shows the result when applying Worley noise only (without adding additionally Perlin noise).

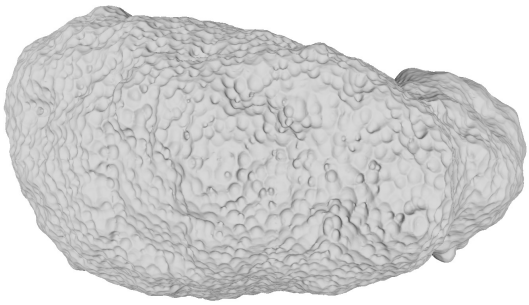

Fig. 7. The rough shape of Itokawa with pure Worley noise and craters appear.

TABLE I
THE NUMBER OF PARAMETERS OF THE SOLUTION DOMAIN IN THE SURFACE DETAIL OPTIMIZATION FOR THE INDIVIDUAL NOISE FUNCTIONS.

Para	Perlin	Simplex	Worley	Gradient
Weight	1	1	1	1
Frequency	1	1	1	0
Octave	1	1	1	0
Amplitude	1	1	1	0
Coords_w	3	3	3	0
Coords_b	3	3	3	0

C. Optimizing Noise Parameters

For simulations in VTBs or the training of autonomous algorithms to steer landing spacecrafts it is essential that the generated asteroids are as realistic as possible. Hence, we do not choose the parameters of our generation methods arbitrary. Manually, selecting the parameters is also not an option because they often behave unexpectedly. Consequently, we add an initial optimization step to adapt the parameters automatically.

Optimization Algorithm

Given some ground truth data, e.g. the detailed model of Itokawa and asteroid databases that provide polygonized rough shapes of asteroids, we can use any optimization algorithm for the parameter adaption provided a good fitness function is available. For several reasons, we decided to apply particle swarm optimization (PSO) [25]. PSO is a stochastic convergence analysis algorithm containing parameter selection and changing. In principle, PSO is a population-based iterative algorithm and the swarm behavior guide the particle in the population to search for globally optimal solutions: The standard PSO algorithm maintains a population of N particles, and each particle defines a potential solution in a D-dimensional solution domain. There exists many factors that influence the convergence property as well as performance of the standard PSO algorithm. In our implementation we rely on fixed parameters according to [26].

The structure of PSO makes it easy to map our parameters to the algorithm. Moreover, PSO is fast and stable with respect to the initial parameter values and it does not require gradient information. We optimize the implicit shape representation and the noise-based (Sec. III-A) surface details (Sec. III-B) individually.

Fitness Function

Choosing a good fitness function to compare the result generated by PSO to the real-world model is essential for the success of the optimization. In our implementation we used a histogram-based shape descriptor presented in [27]. It was developed with the special scope of considering large-scale models with local similarities that typically appear in celestial bodies. Moreover, it is fast and works completely on the GPU, thus, it fits perfectly in our pipeline. However, it is easy to include other fitness functions.

Implicit Shape Optimization

In order to optimize the rough shape defined by the sphere-packing in combination with the metaball approach (Sec. III-A), we simply use the parameters from its description for PSO, namely, the number of spheres, the two tension factors of the potential function (see Equation 3) and the blending parameter from Equation 5. This defined a 4D parameter space. Fig. 4 shows the results of this first optimization stage for Itokawa.

D. Surface Detail Optimization

The surface details are represented by three different noise functions – Perlin, simplex and Worley noise – and additionally, the noise modulation of the gradient of the Perlin and simplex noise. In addition to the individual parameters (like frequency, amplitude, the number of octaves) of the particular noises, we add four weight parameters to control their individual amount. Moreover, we include three parameters to scale the input grid point's 3D coordinate axis and another three parameters to define biases to the axis. Another important parameter is the gradient; it is based on the grid point's shape value and we directly multiply our fractal noise with its gradient value. What's more, we have another three parameters to control the number of perlin noise, simplex noise and worley noise. In total we have 34 parameters to control the possible pattern of the surface details (see Table I).

E. Polygonization

VTBs usually require the 3D objects as polygon meshes instead of implicit representations. However, it is easy to generate such a polygonal mesh by using marching cubes to compute an isosurface. In our recent implementation, we use an hierarchical GPU-based version in order to quickly at generate any required resolution. Please note, also our fitness function described above requires a polygonal representation to generate an appropriate histogram based on the vertices [27]. However, we do not generate a complete high-poly model, but our experiments have shown, that it is sufficient to content with only a subset of around 10% of a full two Million vertex model when using Possion disk sampling [28] .

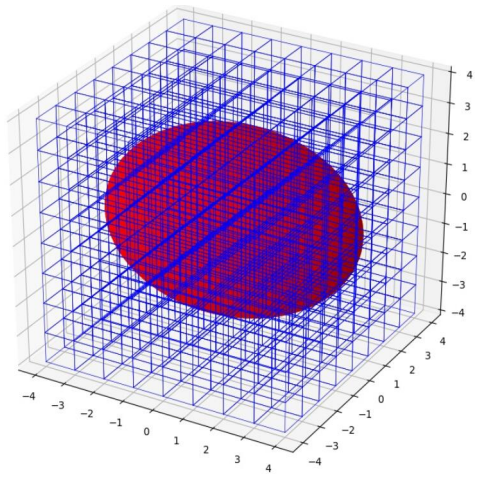

Fig. 8. We use a hierarchical marching cubes algorithm to polygonize our implicit object representation. The blue grid divides the object into several smaller blocks. The marching cubes algorithm is then performed on these smaller blocks individually.

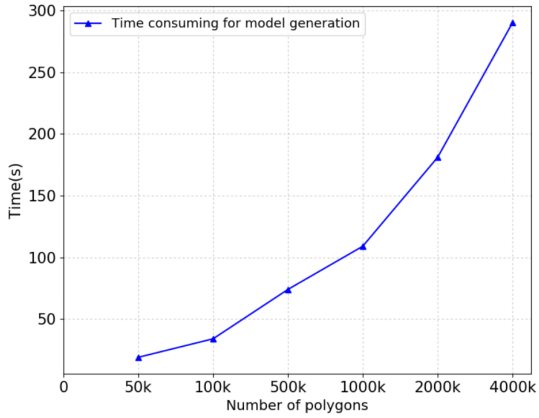

Fig. 9. The time required by our algorithm to generate 3D models of Itokawa in several resolutions.

IV. RESULTS AND DISCUSSIONS

We have implemented AstroGen in C++ and CUDA, including the implicit generation and the optimization algorithm. The computation of the noise values is implemented using computer shaders and marching cubes algorithm relies on geometry shaders in OpenGL. We performed our experiments on a machine with Intel Core i7 8-core processor with 8GB of RAM and an Nvidia GTX1080Ti. We have evaluated the performance as well as the quality in two different test scenarios. The basis is the optimization of all parameters for an available high poly-asteroid model of Itokawa. We set the problem space to $D = 34$ according to Section III-D. The population size of PSO was set to $N = 20$ and we allowed at most 100 generations.

First, we investigated the performance to generate 3D objects from our implicit representation with respect to the polygon count. Fig. 9 shows the mean average computation time for the specific resolutions. The time increases almost linear with an increasing number of polygons. Please note, that our current marching cubes implementation is not yet fully optimized. In the future, we hope to improve the performance e.g. by additionally applying an octree to accelerate the generation time. The training phase took approximatively 8 hours for the implicit shape representation and 100 hours for the noise-based surface features. Most time was spend on the generation of the high-resolution model (90%), while the computation of the fitness function required 10% of the overall time.

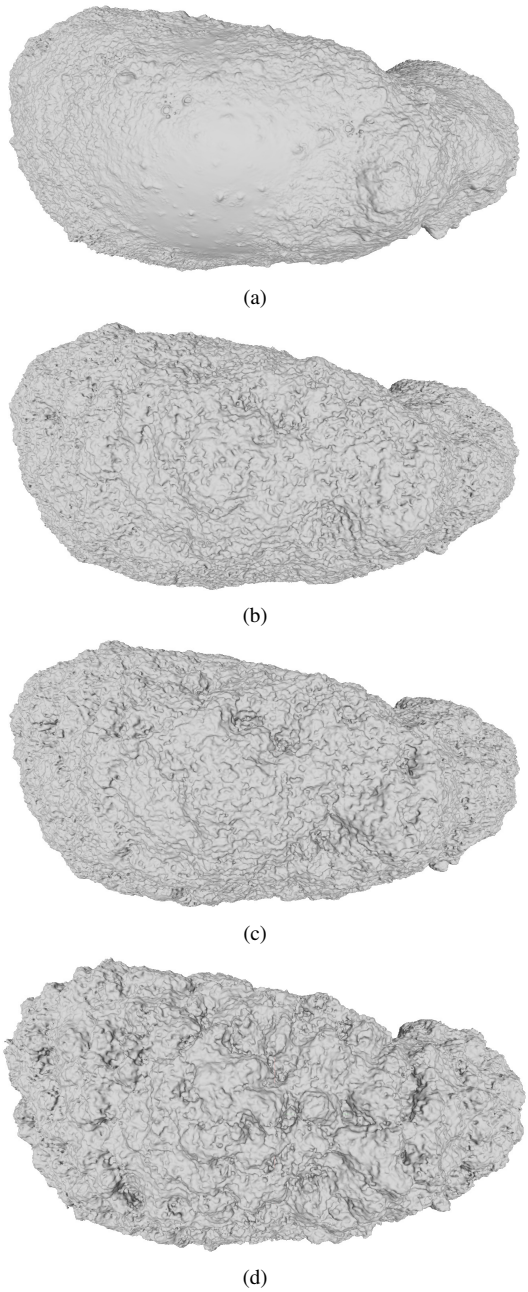

Fig. 10. (a) The original model of the asteroid Itokawa (1,780k vertices). Generated asteroid models with a similarity of (b) 95% (1,986k vertices), (c) 90% (2,173k vertices) and (d) 85% (2,335k vertices).

In our first test scenario, we applied our method to automatically generate similar asteroids with small variations that are below a $\delta < 20\%$ with respect to the fitness function.

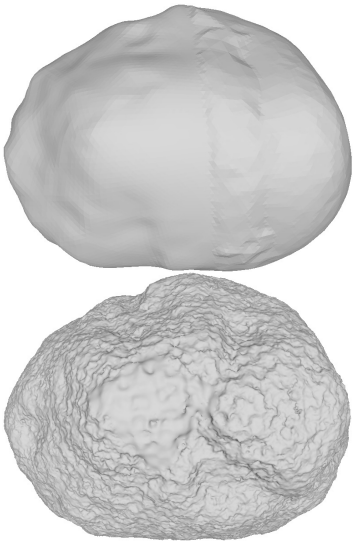

Fig. 11. The original low-resolution Stein model (10k vertices) and our automatically generated model with surface details (710k vertices).

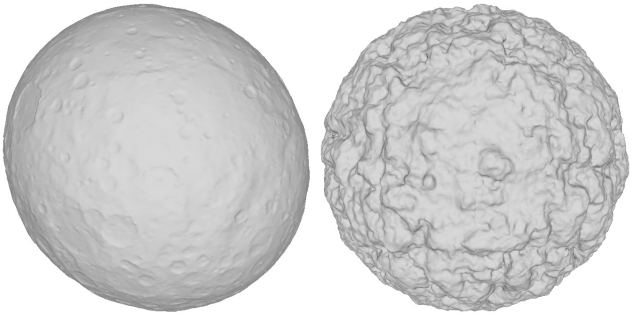

Fig. 12. The original low-resolution Ceres model (128k vertices) and our automatically generated model with surface details (1,063k vertices).

Fig. 10 shows the original model and some of the results[2]. Fig. 10(b) exhibits most similarity with the original Itokawa model, followed by a rapidly decreasing in similarity but two different terrain patterns in Fig. 10(c) and Fig. 10(d).

In our second scenario, we transferred the parameter set for the surface details that were trained with the Itokawa model to other (low-poly) basic shapes from an asteroid database that were never explored with a spacecraft. Fig. 11, 12 and 13 show the results for Stein, Ceres and Lutetia, respectively.

V. CONCLUSIONS AND FUTURE WORKS

We have presented the first fully automatic method to generate highly detailed realistic models of small celestial bodies. The main idea is to combine a two-tier approach of implicit shape representation and different noise functions with an optimization algorithm. This enables us to "learn" from real world or hand crafted models and automatically generate an almost infinite number of small (or larger) variations. Even more, our results have shown that we can apply the learned parameters also to other, e.g. low-poly, models to generate a similar surface structure. AstroGen runs completely massively parallel on the GPU which indicates a high performance. As the trend in future space exploration tends to focus on objects in deep space, the importance of autonomy increases on-board of spacecraft. Hence, AstroGen could be an important step to enable decision making in virtual testbed by considering different scenarios and for the training of autonomous algorithms in space crafts. Moreover, our sphere based implicit surface be can easily extended to support a mascons-based model to simulate the gravity of asteroid more accurately, even within the Brouillon-sphere.

However, AstroGen also offers interesting avenues for future work. For instance, we want to investigate heteromorphic shape reproduction and improve the quality of the mesh in general. For instance, we want to consider different basic shapes, instead of spheres, for the implicit surface reconstruction with metaballs such as cubes, tori or ellipsoids. This can be further combined with a tree-like structure similar to BlobTrees [29]. For the surface details, we want to investigate other noise functions and more parameters, but also thermal erosion, hydraulic erosion algorithms are interesting [30]. This could improve the naturalness of AstroGen. Finally, we want to improve the mesh generation, for instance by adopting dual marching cube [31] to enhance the visual fidelity of the isosurfaces.

[2]For more results please visit https://github.com/XZ-CG/asteroid-result

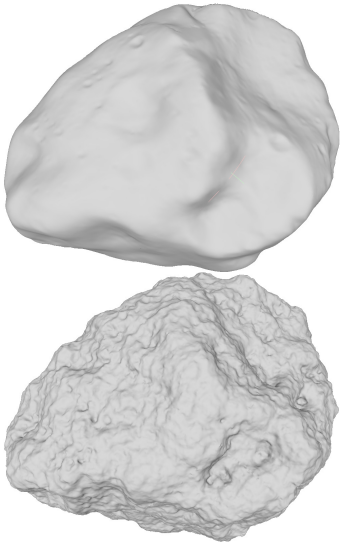

Fig. 13. The original low-resolution Lutetia model (122k vertices) and our automatically generated model with surface details (778k vertices).

REFERENCES

[1] P. Lange, R. Weller, and G. Zachmann, "Multi agent system optimization in virtual vehicle testbeds." in *SimuTools*, 2015, pp. 79–88.

[2] J. Farnham, "Shape model of asteroid 2867 steins, ro-a-osinac/osiwac-5-steins-shape-v1.0," NASA Planetary Data System, 2013.

[3] P. Thomas, J. W. Parker, L. McFadden, C. T. Russell, S. Stern, M. Sykes, and E. Young, "Differentiation of the asteroid ceres as revealed by its shape," *Nature*, vol. 437, no. 7056, p. 224, 2005.

[4] H. Sierks, P. Lamy, C. Barbieri, D. Koschny, H. Rickman, R. Rodrigo, M. F. AHearn, F. Angrilli, M. A. Barucci, J.-L. Bertaux *et al.*, "Images of asteroid 21 lutetia: a remnant planetesimal from the early solar system," *science*, vol. 334, no. 6055, pp. 487–490, 2011.

[5] J.-D. Génevaux, É. Galin, E. Guérin, A. Peytavie, and B. Benes, "Terrain generation using procedural models based on hydrology," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 143, 2013.
[6] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions," in *Computer Graphics Forum*, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2579–2600.
[7] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A survey on procedural modelling for virtual worlds," in *Computer Graphics Forum*, vol. 33, no. 6. Wiley Online Library, 2014, pp. 31–50.
[8] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.
[9] ——, "Improving noise," in *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3. ACM, 2002, pp. 681–682.
[10] D. Ebert, F. Musgrave, D. Peachey, K. Perlin, S. Worley, W. Mark, and J. Hart, *Texturing and Modeling: A Procedural Approach: Third Edition*. United States: Elsevier Inc., 2003.
[11] H. Zhou, J. Sun, G. Turk, and J. M. Rehg, "Terrain synthesis from digital elevation models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 4, pp. 834–848, July/August 2007.
[12] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Proceedings of the 2010 workshop on procedural content generation in games*. ACM, 2010, p. 3.
[13] I. Parberry, "Designer worlds: Procedural generation of infinite terrain from real-world elevation data," *Journal of Computer Graphics Techniques*, vol. 3, no. 1, 2014.
[14] J.-D. Génevaux, E. Galin, A. Peytavie, E. Guérin, C. Briquet, F. Grosbellet, and B. Benes, "Terrain modelling from feature primitives," in *Computer Graphics Forum*, vol. 34, no. 6. Wiley Online Library, 2015, pp. 198–210.
[15] E. Guérin, J. Digne, E. Galin, and A. Peytavie, "Sparse representation of terrains for procedural modeling," in *Computer Graphics Forum*, vol. 35, no. 2. Wiley Online Library, 2016, pp. 177–187.
[16] J. Ďurech, V. Sidorin, and M. Kaasalainen, "Damit: a database of asteroid models." *Astronomy & Astrophysics*, vol. 513, no. A46, 2010. [Online]. Available: http://www.aanda.org/articles/aa/abs/2010/05/aa12693-09/aa12693-09.html
[17] B. Wyvill, A. Guy, and E. Galin, "Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system," in *Computer Graphics Forum*, vol. 18, no. 2. Wiley Online Library, 1999, pp. 149–158.
[18] J. F. Blinn, "A generalization of algebraic surface drawing," *ACM transactions on graphics (TOG)*, vol. 1, no. 3, pp. 235–256, 1982.
[19] J. Teuber, R. Weller, G. Zachmann, and S. Guthe, "Fast sphere packings with adaptive grids on the gpu," *In GI AR/VRWorkshop (Würzburg, Germany*, vol. 4, 2013.
[20] H. J. Miller, "Tobler's first law and spatial analysis," *Annals of the Association of American Geographers*, vol. 94, no. 2, pp. 284–289, 2004.
[21] H. Nguyen, *Gpu gems 3*. Addison-Wesley Professional, 2007.
[22] S. Worley, "A cellular texture basis function," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 291–294.
[23] J. L. Patricio Gonzalez Vivo, "The book of shaders," https://thebookofshaders.com/, 2015.
[24] S. Gustavson, "cellular noise in glsl: Implementation notes," 2011.
[25] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*. IEEE, 1995, pp. 39–43.
[26] N. R. Samal, A. Konar, S. Das, and A. Abraham, "A closed loop stability analysis and parameter selection of the particle swarm optimization dynamics for faster convergence," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*. IEEE, 2007, pp. 1769–1776.
[27] X. Li, P. Lange, R. Weller, and G. Zachmann, "Invariant local shape descriptors: classification of large-scale shapes with local dissimilarities," in *Proceedings of the Computer Graphics International Conference*. ACM, 2017, p. 9.
[28] M. S. Ebeida, S. A. Mitchell, A. A. Davidson, A. Patney, P. M. Knupp, and J. D. Owens, "Efficient and good delaunay meshes from random points," *Computer-Aided Design*, vol. 43, no. 11, pp. 1506–1515, 2011.
[29] E. P. De Groot, *Blobtree modelling*. University of Calgary, 2008.
[30] A. Lagae, S. Lefebvre, R. Cook, T. Derose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "State of the art in procedural noise functions," in *EG 2010-State of the Art Reports*. The Eurographics Association, 2010.
[31] S. Schaefer and J. Warren, "Dual marching cubes: Primal contouring of dual grids," in *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*. IEEE, 2004, pp. 70–76.