Signed distance function generation using algebraic point set surfaces

Murray Cutforth[a,*]

[a]*Centre for Scientific Computing, Cambridge University, Cambridge, CB3 0HE, UK*

Abstract

In this document the algebraic point set surface (APSS) method is applied to the problem of generating a signed distance function from a triangulated three-dimensional surface. Initial results from this approach are promising, as it is extremely fast, easy to implement, and can robustly handle bad input data (such as non-closed surfaces). As described here the method is prone to generating spurious artifacts in the zero-level contour of the signed distance function, but there is large scope for development and improvement to the method.

1. Introduction

Algebraic point set surfaces were developed recently in [2] and [1] in order to fit a surface to a set of points with normals. This work has previously found application in reconstructing surfaces from point clouds, such as those obtained from range-based laser scanners, or in point-based computer graphics. They offer a robust, accurate, and cheap method of calculating a scalar function whose zero level set implicitly defines the surface.

The APSS method takes a cloud of points with normals as input, and returns a function which can be evaluated anywhere in $\mathbf{x} \in \mathbb{R}^d$, returning the approximate signed distance function. The APSS approach is well established, but has not previously been used in the context of generating a signed distance function from a triangulated surface.

2. Algebraic point set surfaces

The APSS approach described in [2] and [1] defines a surface as the zero level set of the scalar function defining an algebraic sphere in d spatial dimensions. An algebraic sphere is defined as:

$$s(\mathbf{x}) = (1, \mathbf{x}^T, \mathbf{x}^T\mathbf{x})\,\mathbf{u}(\mathbf{x}) = 0, \quad \mathbf{x} \in \mathbb{R}^d, \quad \mathbf{u} \in \mathbb{R}^{d+2} \tag{1}$$

and we define the surface $\mathcal{S}$ as:

$$\mathcal{S} = \{\mathbf{x} \mid s(\mathbf{x}) = 0\}. \tag{2}$$

The fitted algebraic sphere moves continuously in space as $\mathbf{x}$ changes, and approaches a planar fit in the degenerate case of a single oriented point.

We fit a surface to the set of points $P = \{\mathbf{p}_i \in \mathbb{R}^d\}$ with unit normals $\mathbf{n}_i$ and local spacing r_i. The relative contribution of each point to the surface fitting is given by the adaptive weighting scheme

$$w_i(\mathbf{x}) = \phi\left(\frac{\|\mathbf{x} - \mathbf{p}_i\|}{r_i\, h}\right), \tag{3}$$

where h is a parameter of the scheme which controls the amount of smoothing of the surface, and

$$\phi(t) = \begin{cases} (1 - x^2)^4 & \text{if } x < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

We want to find the coefficients $\mathbf{u}$ of the algebraic sphere such two objectives are achieved. Firstly, we aim to minimise the differences between the gradient of the function $s(\mathbf{x})$ and the normal $\mathbf{n}_i$ at each $\mathbf{p}_i$:

$$\sum_i w_i \|\nabla s(\mathbf{p}_i) - \mathbf{n}_i\|^2. \tag{5}$$

Secondly, we aim to minimise the value of $s(\mathbf{x})$ at each $\mathbf{p}_i$:

$$\sum_i w_i (s(\mathbf{p}_i))^2. \tag{6}$$

It turns out that we can obtain a closed form solution for $\mathbf{u}(\mathbf{x})$ using some relatively straightforward linear algebra. Following the novel approach in [1] we minimise the position and derivative constraints separately.

*Corresponding author.
Email address: mcc74@cam.ac.uk (Murray Cutforth)

2.1. Derivative constraints

The derivative constraints aim to match the gradient of the surface to the normals provided at each input point. This ensures that the interior and exterior regions of $s(\mathbf{x})$ are consistently signed, and since the normals have unit length, the function $s(\mathbf{x})$ is approximately a signed distance function close to the surface $\mathcal{S}$. The derivative constraints are sufficient to determine all coefficients of $\mathbf{u}$ except for the first coefficient, which we denote by u_0. We begin by expanding using the definition of the gradient:

$$\mathbf{u}(\mathbf{x}) = \arg\min_{\mathbf{u}} \sum_i w_i \parallel (0, \mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_d, 2\mathbf{p}_i)\mathbf{u} - \mathbf{n}_i \parallel^2, \tag{7}$$

where we have the orthonormal basis vectors $\mathbf{e}_i$. Using this we re-write equation 5 as the linear system:

$$\mathbf{u}(\mathbf{x}) = \arg\min_{\mathbf{u}} \|\mathbf{D}\mathbf{u}\text{'} - \mathbf{b}\|^2, \tag{8}$$

where $\mathbf{D}$ is the $(Nd) \times (d+1)$ matrix:

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}\text{'}_1 \\ \mathbf{D}\text{'}_2 \\ \vdots \\ \mathbf{D}\text{'}_N \end{pmatrix}, \quad \mathbf{D}\text{'}_i = \begin{pmatrix} \sqrt{w_i}\mathbf{e}_1^T & 2\sqrt{w_i}(\mathbf{p}_i)_1 \\ \sqrt{w_i}\mathbf{e}_2^T & 2\sqrt{w_i}(\mathbf{p}_i)_2 \\ \vdots & \vdots \\ \sqrt{w_i}\mathbf{e}_d^T & 2\sqrt{w_i}(\mathbf{p}_i)_d \end{pmatrix}, \tag{9}$$

and $\mathbf{b} \in \mathbb{R}^{Nd}$ is the vector

$$\mathbf{b} = \begin{pmatrix} \mathbf{b}\text{'}_1 \\ \mathbf{b}\text{'}_2 \\ \vdots \\ \mathbf{b}\text{'}_N \end{pmatrix}, \quad \mathbf{b}\text{'}_i = \begin{pmatrix} \sqrt{w_i}(\mathbf{n}_i)_1 \\ \sqrt{w_i}(\mathbf{n}_i)_2 \\ \vdots \\ \sqrt{w_i}(\mathbf{n}_i)_d \end{pmatrix} \tag{10}$$

and $\mathbf{u}\text{'} \in \mathbb{R}^{d+1}$ is the solution vector

$$\mathbf{u}\text{'} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \\ u_{d+1} \end{pmatrix}. \tag{11}$$

Note that this vector has dimension one less than the original solution vector $\mathbf{u}$ in equation 7, due to the leading column of zeroes in the matrix $\nabla(1, \mathbf{x}^T, \mathbf{x}^T\mathbf{x}) = (0, \mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_d, 2\mathbf{x})$. In general, we have that $Nd > d+1$ so the system in equation 8 is overdetermined. Gauss and Legendre discovered that the solution $\mathbf{u}\text{'}$ to this minimisation problem is the solution to the $(d+1) \times (d+1)$ system

$$\mathbf{D}^T\mathbf{D}\mathbf{u}\text{'} = \mathbf{D}^T\mathbf{b}, \tag{12}$$

and that the matrix $\mathbf{D}^T\mathbf{D}$ is invertible if the columns of $\mathbf{D}$ are linearly independent. In our case, the first d columns of the matrix are constructed from the orthonormal basis vectors, and are therefore linearly independent. The $(d+1)$-th column is constructed using the components of the input point set coordinates $\mathbf{p}_i$, and is also linearly independent except for in special cases. The matrix $\mathbf{D}^T\mathbf{D}$ takes the following form:

$$\mathbf{D}^T\mathbf{D} = \begin{pmatrix} \mathbf{I} & 2\sum_i w_i\mathbf{p}_i \\ 2\sum_i w_i\mathbf{p}_i^T & 4\sum_i w_i\|\mathbf{p}_i\| \end{pmatrix}, \tag{13}$$

where $\mathbf{I}$ is the $d \times d$ identity matrix. Amazingly, we can invert this matrix by hand in an arbitrary number of dimensions, using the vector inner product to simplify the results. This yields the solution:

$$u_{d+1} = \frac{1}{2}\frac{(\sum_i w_i)\left(\sum_i w_i\mathbf{n}_i^T\mathbf{p}_i\right) - \left(\sum_i w_i\mathbf{n}_i^T\right)(\sum_i w_i\mathbf{p}_i)}{(\sum_i w_i)\left(\sum_i w_i\mathbf{p}_i^T\mathbf{p}_i\right) - \left(\sum_i w_i\mathbf{p}_i^T\right)(\sum_i w_i\mathbf{p}_i)}, \tag{14}$$

$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix} = \frac{\sum_i w_i\mathbf{n}_i - 2u_{d+1}\sum_i w_i\mathbf{p}_i}{(\sum_i w_i)}. \tag{15}$$

2.2. Position constraints

It now only remains to find the scalar u_0, such that the algebraic distance is minimised:

$$u_0 = \arg\min_{u_0} \sum_i w_i \left(s(\mathbf{p}_i)\right)^2. \tag{16}$$

By substituting the definition of $s(\mathbf{x})$, we obtain

$$u_0 = \arg\min_{u_0} \left\| \begin{matrix} \sqrt{w_1}\left(1, \mathbf{p}_1^T, \mathbf{p}_1^T\mathbf{p}_1\right)\mathbf{u} \\ \sqrt{w_2}\left(1, \mathbf{p}_2^T, \mathbf{p}_2^T\mathbf{p}_2\right)\mathbf{u} \\ \vdots \\ \sqrt{w_N}\left(1, \mathbf{p}_N^T, \mathbf{p}_N^T\mathbf{p}_N\right)\mathbf{u} \end{matrix} \right\|^2, \tag{17}$$

which can be written in linear least squares form as

$$u_0 = \arg\min_{u_0} \left\| \begin{pmatrix} \sqrt{w_1} \\ \sqrt{w_2} \\ \vdots \\ \sqrt{w_N} \end{pmatrix} u_0 - \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix} \right\|^2, \tag{18}$$

where

$$b_i = -\sqrt{w_i}\left((u_1, u_2, \ldots, u_d)\mathbf{p}_i + u_{d+1}\mathbf{p}_i^T\mathbf{p}_i\right). \tag{19}$$

This problem can be solved trivially as the solution of the 1×1 linear system

$$\left(\sum_i w_i\right) u_0 = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}^T \sum_i w_i \mathbf{p}_i - u_{d+1} \sum_i w_i \mathbf{p}_i^T \mathbf{p}_i. \quad (20)$$

2.3. Signed distance function estimation

Using the closed form solution for $\mathbf{u}(\mathbf{x})$ derived above, given a set of input points, it is possible to evaluate the value of the algebraic surface $s(\mathbf{x})$ at any coordinate with just one iteration over all the points with nonzero weighting. However although the algebraic surface is consistently signed with respect to interior/exterior regions, it is not a signed distance function. Fortunately we can obtain this very easily by converting the equation of the fitted algebraic sphere at $\mathbf{x}$ into explicit form: by completing the square on equation 1 we obtain

$$(\mathbf{x} - \mathbf{c})^2 = \mathbf{c}^T\mathbf{c} - \frac{u_0}{u_{d+1}} \quad (21)$$

where

$$\mathbf{c} = \frac{-1}{2u_{d+1}} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}. \quad (22)$$

From this form we see that the centre of the fitted circle is at $\mathbf{c}$, with a radius of $r = \sqrt{\mathbf{c}^T\mathbf{c} - \frac{u_0}{u_{d+1}}}$. As well as giving a computationally-free estimate of the surface curvature at that point, this gives the signed distance ϕ as

$$\phi(\mathbf{x}) = \sqrt{\mathbf{c}^T\mathbf{x}} - r. \quad (23)$$

2.4. Remarks on the APSS method

The APSS algorithm has been implemented such that it is independent of dimension. The full single-thread C++ programe comes to approximately 500 lines of code- it is very straightforward.

There are two areas for improvement which have been identified in the use of this algorithm. The first is the weight function- at present the method is able to "see" the other side of thin surfaces since the weight function is rather wide. This is solvable with the use of a narrower weight function, or even by restricting the reconstruction to only use the n-nearest particles. The second issue is connected to this- it is the question of what to do in the case where no input points are within distance h of the sample position. In this case the weighting on every point evaluates to zero. This is dealt with by iteratively increasing the value of h until at least one of the w_i becomes nonzero. The increment on h does affect the form of the scalar function $s(\mathbf{x})$. At present, h is doubled with each iteration. A smaller increment may be slightly less efficient but would be expected to produce smoother isocontours of ϕ.

3. Point cloud generation

The APSS method takes an oriented point cloud as input. Generating evenly-distributed points over the triangulated surface is critical to the robust operation of the APSS method. Assuming that the total number of points is a user-defined parameter, N, the average point density is computed as $\frac{N}{A}$, where A is the total surface area of the triangulation. The following procedure is then applied to every triangle on the surface, where the triangle vertices are denoted by $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$.

1. Compute area of current triangle

$$a_c = \frac{1}{2}|(\mathbf{x}_3 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_2)|. \quad (24)$$

2. Populate this triangle with n_c markers, where:

$$n_c = \text{ceil}\left(a_c \cdot \frac{N}{A}\right). \quad (25)$$

3. Seed n_c random points at positions

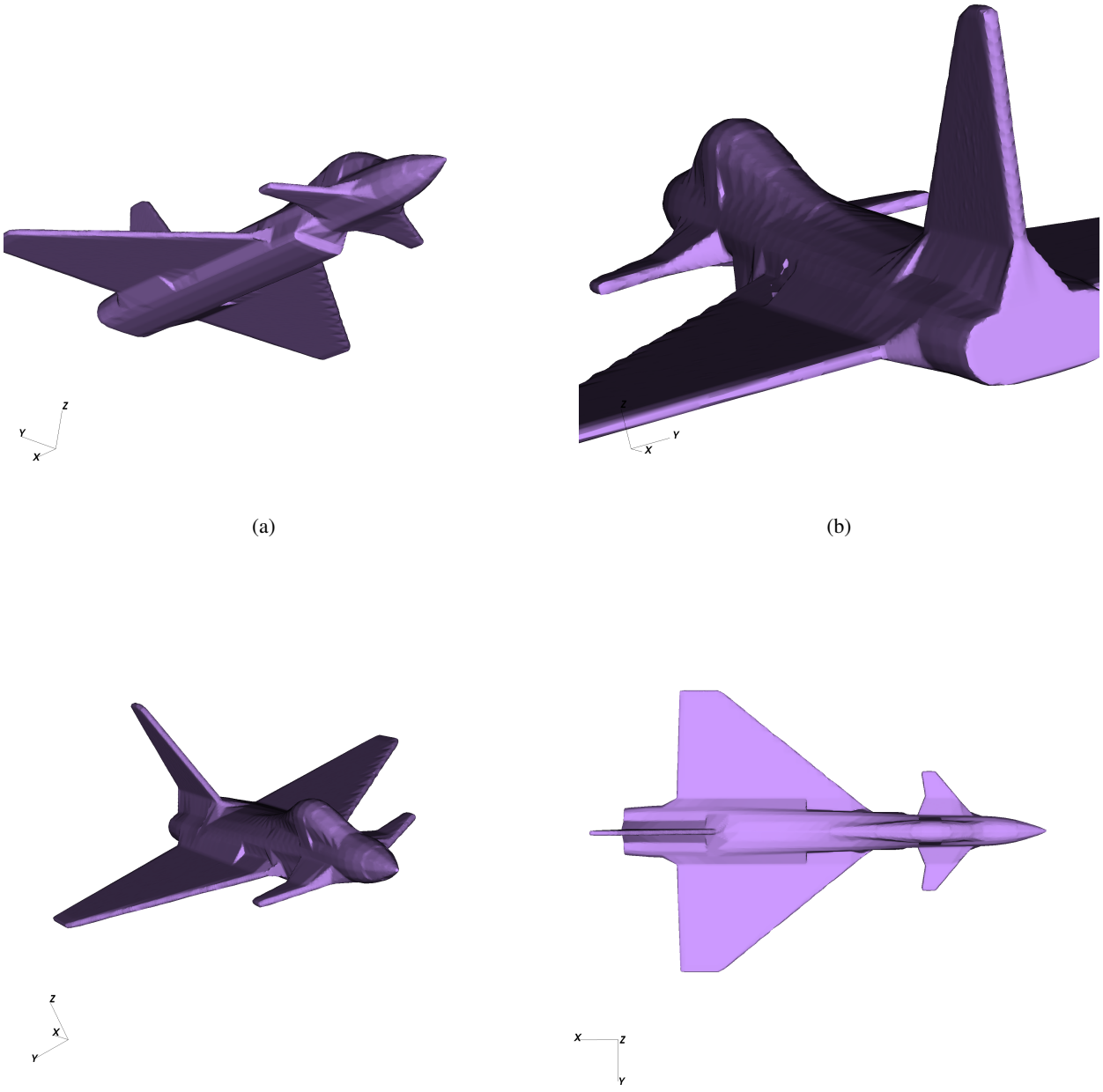
$$\mathbf{x}_1 + a(\mathbf{x}_2 - \mathbf{x}_1) + b(\mathbf{x}_3 - \mathbf{x}_1), \quad (26)$$

where a, b are random numbers uniformly distributed over $[0, 1]$ and points satisfying $a + b > 1$ are discarded. All points are stored with the normal defined by the triangle.

4. Preliminary results

In this section the method is applied to generate signed distance functions from a variety of triangulated surfaces defined by the STL file format. The zero level sets are plotted using the Visit software, which reconstructs them from ϕ using the marching cubes algorithm. Results are presented for qualitative comparison only.

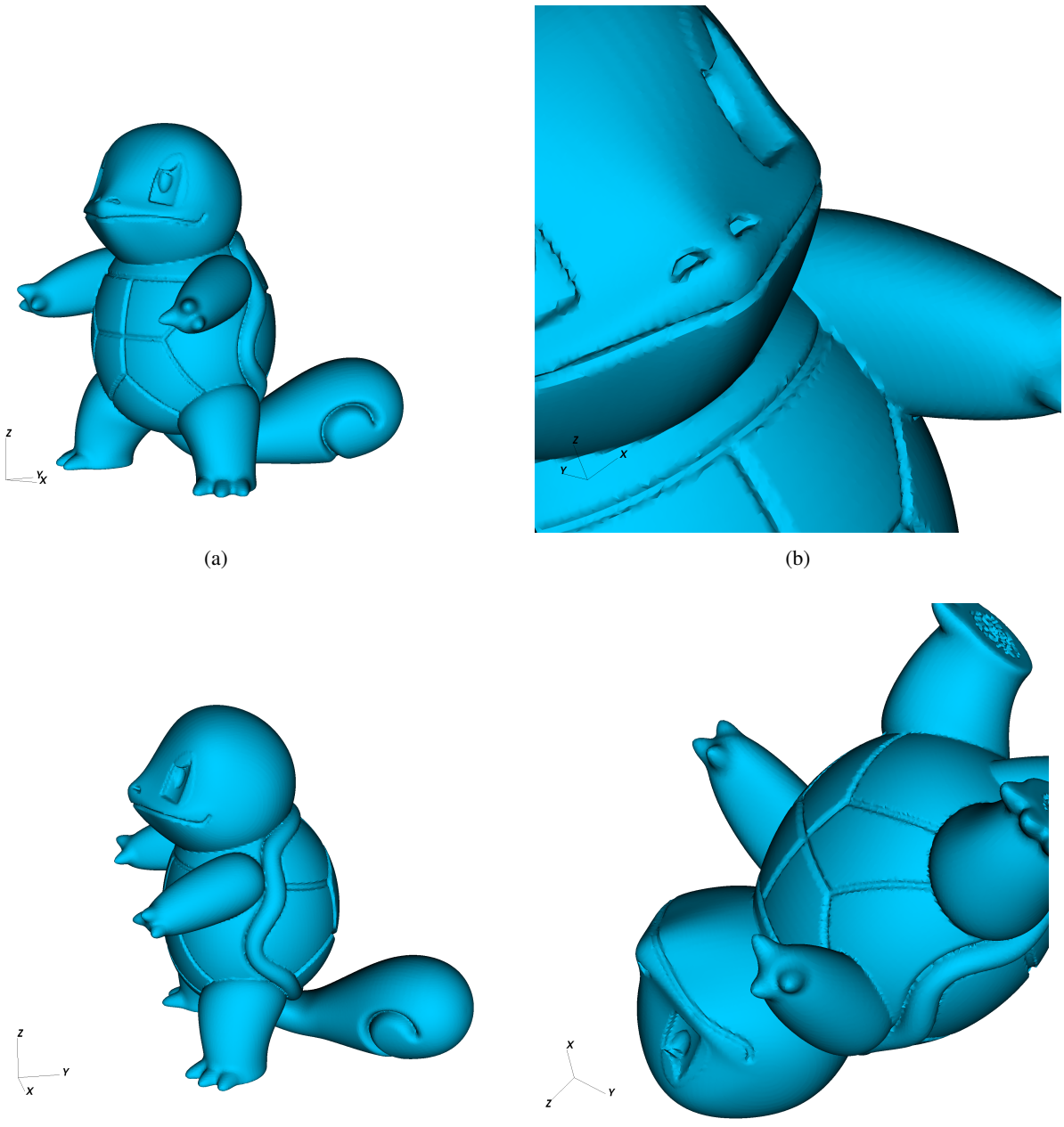
Note that in the current implementation, the APSS is only called in a narrow band around the surface to generate ϕ. The remainder of the domain is populated with appropriately-signed values using a single fast-sweeping-style iteration. Also, none of the STL files have been passed through any cleanup software.



(a) (b)

(c) (d)

Figure 1: The zero-level set of the signed distance function for the Eurofighter STL sampled on a 400^3 mesh. There were 4×10^6 marker particles seeded over the triangulated surface. The code ran in ~5 minutes. In (b) a small artefact is visible over the left wing- this is a small region erroneously computed to be inside the plane due to part of one triangle protruding from the surface.



(a) (b)

(c) (d)

Figure 2: The zero-level set of the signed distance function for the Squirtle STL sampled on a $240 \times 320 \times 320$ mesh. There were 4×10^6 marker particles seeded over the triangulated surface. The code ran in ~2 minutes.

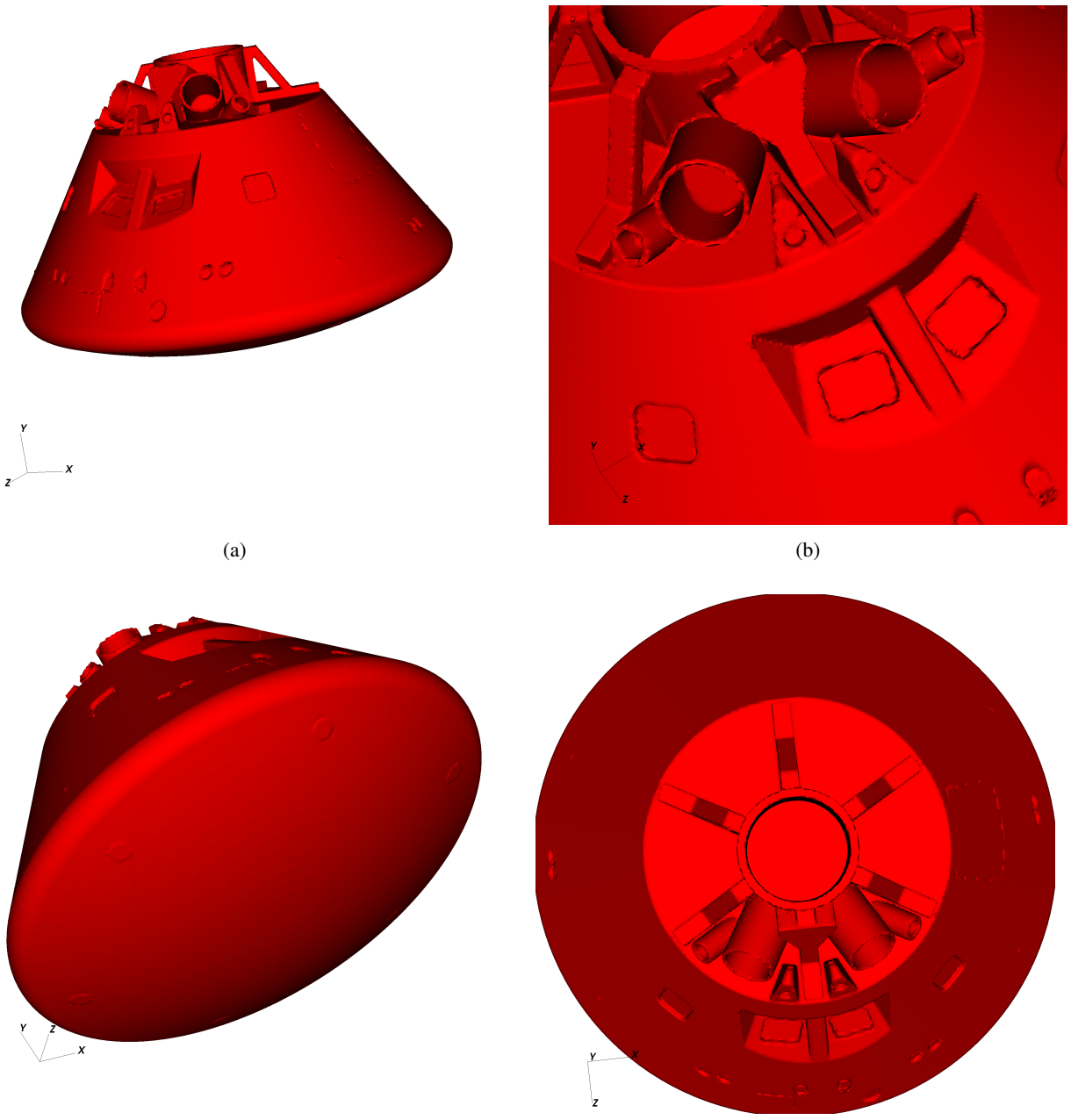


Figure 3: The zero-level set of the signed distance function for the Orion capsule STL sampled on a 300^3 mesh. There were 25×10^6 marker particles seeded over the triangulated surface. The code ran in ~10 minutes.

5. Conclusions and further work

This study has shown that it is possible to use the APSS methodology to generate a signed distance function from surface triangulation data. The method is fast, robust, and a single thread three-dimensional implementation requires ~500 lines of C++ code. There is much scope for further work to develop the capability of the method in handling 'bad' input data, and the method as implemented currently is prone to confusion when the length scale of surface features is much smaller than the length scale of cells. The following points describe how the work could be extended.

- The evaluation of ϕ throughout the grid using the APSS method is an embarrassingly parallel problem. This method could be massively accelerated with relatively little effort, and should scale very well.
- The weight function and the choice of which marker points to include in the computation of $\phi(\mathbf{x})$ are critical. It would be interesting to experiment with tighter weight functions- even up to the limit that every sample point is computed from the single nearest marker particle (in this degenerate case the algebraic sphere becomes a plane).

References

[1] Gal Guennebaud, Marcel Germann, and Markus Gross. Dynamic Sampling and Rendering of Algebraic Point Set Surfaces. *Computer Graphics Forum*, 27(2):653–662, 4 2008.

[2] Gal Guennebaud and Markus Gross. Algebraic point set surfaces. *ACM Transactions on Graphics*, 26(99):23, 7 2007.