Reconstructing a shape from a point cloud

In the last chapter we learned about surfaces, their normals, and how they are represented on the computer: as triangle meshes. This chapter deals with one way of creating such surfaces: by acquiring them from the real world via cameras or laser scanning.

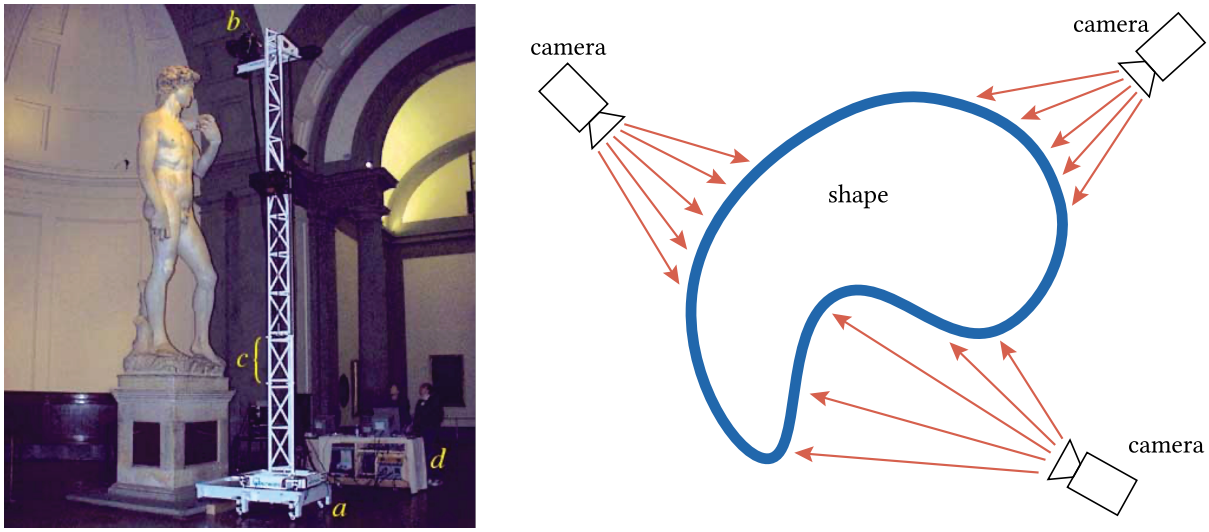

Figure 1: Acquiring point samples on a real-world shape by laser scanning *(left)* [4]. A schematic overview of the laser scanning process *(right)*.

An optical sensor can acquire information about a surface by scanning every visible point and learning two properties for every angle it scanned in:

- the surface's *distance* at that angle;
- the surface's *normal* at that angle

There are many ways of obtaining this information. A laser, for example, can measure travel time of a beam, and can use reflectance to measure normals.[1] This process is schematically represented in Figure 1, which also features a real-world example.

Point clouds

An optical scanner as in Figure 1 does not produce a triangle mesh. Instead, it produces a *point cloud*.

Definition 1: Point Cloud

A *point cloud* is a surface representation consisting of a matrix $\mathrm{P} \in \mathbb{R}^{n \times d}$ of point positions as well as a matrix $\mathrm{N} \in \mathbb{R}^{m \times d}$ of normals ($d = 2$ is the dimension of the surrounding space). each row of P is a point and corresponds to a row of N.

P and N approximate the set of points Ω of a continuous surface and its normals.

Figure 2 contrasts point clouds with smooth surfaces and triangle meshes.

[1] This is, of course, a simplification. There are many ways to obtain surface information from the real world with optical sensors, and they are all slightly different. This model is an abstraction – this is not an optics course.

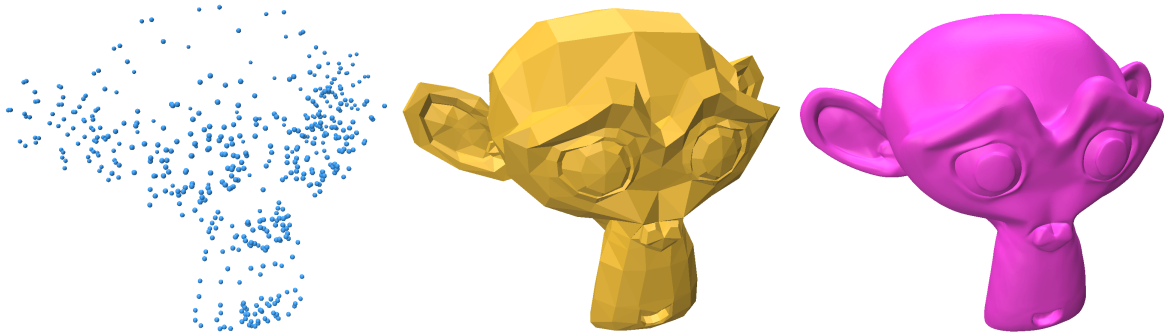

Figure 2: Corresponding point cloud *(left)*, triangle mesh *(middle)*, and smooth surface *(right)*.

A point cloud gives us less information than a triangle mesh. For example, we don't really know where exactly the boundary between inside and outside are away from the points. This also a problem when trying to draw a surface. A triangle mesh is easy to draw as a continuous surface, but a point cloud is not – what happens between the points?

The method that is the subject of this chapter, *Poisson Surface Reconstruction* (PSR) [3], is a way to convert point clouds into triangle meshes.

Volumetric representation

Before we can learn about how to transform point clouds into triangle meshes, we must first learn another surface representation which will be used by the algorithm: *volumetric representation*.

Definition 2: Indicator function and Volume

The function $\chi : \mathbb{R}^n \to [0,1]$ is an indicator function of the surface Ω if

$$\chi(x) = \begin{cases} 1 \text{ if } x \text{ inside } \Omega \\ 0 \text{ if } x \text{ outside } \Omega \end{cases}. \tag{1}$$

S is the volume bounded by Ω, i.e.,

$$\chi(x) = \begin{cases} 1 \text{ if } x \in S \\ 0 \text{ if } x \notin S \end{cases}. \tag{2}$$

The indicator function from Definition 2 completely represents the surface Ω. Figure 3 shows an overview.

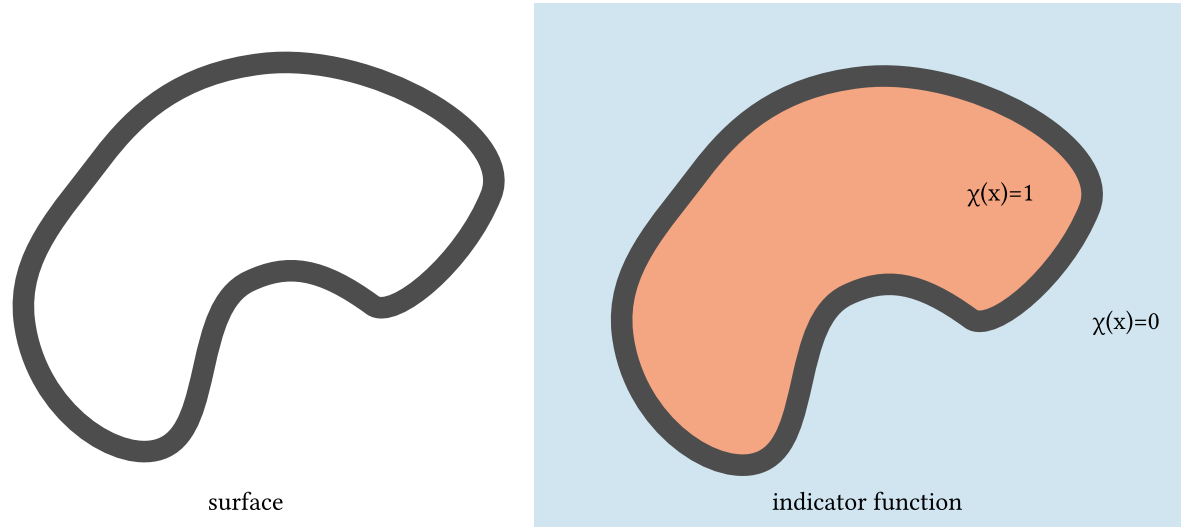

Figure 3: A surface *(left)* and its corresponding indicator function *(right)*.

"Something to think about" 1: Indicator functions and surface points

What happens if $x \in \Omega$, i.e., the point x is exactly on the surface Ω?

One limitation of volumetric representations is that they can only model objects with a well-defined inside and outside. Thus, PSR (as described in this chapter) will only work on *closed* surfaces.

Poisson Surface Reconstruction

With the tools of point clouds and volumetric representation in hand, we can now describe the Poisson Surface Reconstruction (PSR) algorithm [3]. PSR first converts a point cloud into an indicator function, and then constructs a triangle mesh from the indicator function. We will look at both steps separately.

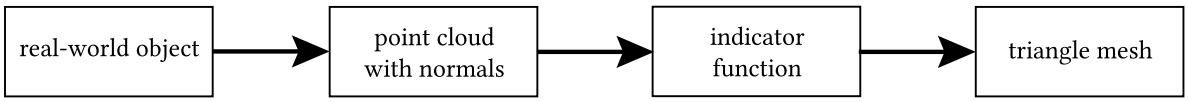

Figure 4: An overview of all steps of the Poisson Surface Reconstruction (PSR) method.

Point cloud to indicator function

One tempting way to use the point cloud `P,N` to construct an indicator function γ might be to just try to find a function that has the right value $\gamma(x)$ at all the points of the point cloud P. But, as alluded to in "Something to think about" 1, it's not that easy. The surface Ω is actually the only point where we do not have a good definition of γ.

Instead, PSR uses the gradient information of the indicator function γ. The gradient $\nabla\gamma(x)$ points towards the direction of the largest change of γ at point x, the the magnitude of $\nabla\gamma(x)$ is the amount of change. $\nabla\gamma(x)$ is zero almost everywhere, since γ is constant almost everywhere. $\nabla\gamma(x)$ is nonzero exactly on the surface Ω itself, where it corresponds to the normals of the surface. The gradient is technically infinite there, but it points in the right direction. This situation is illustrated in Figure 5.

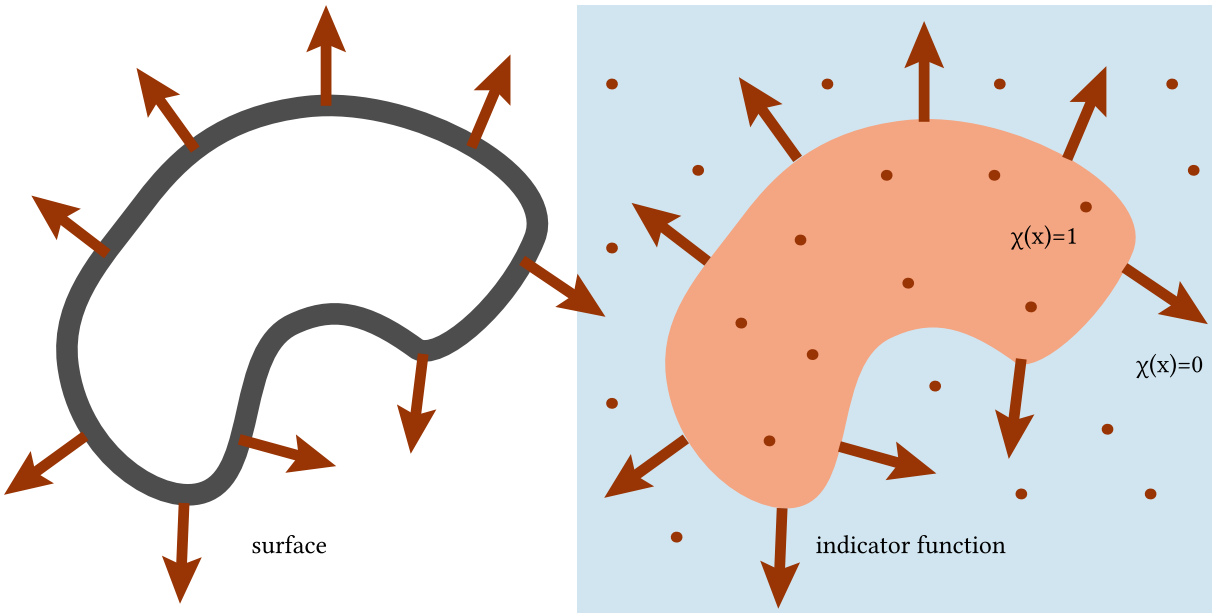

Figure 5: The normals of a surface *(left)* correspond to the gradient of the indicator function, which is zero almost anywhere, and infinite on the surface ifself *(right)*.

We still have two problems: the gradient $\nabla\gamma(x)$ is infinite, and $\nabla\gamma(x)$ is not well-defined, since it is only nonzero on the tiny segment Ω, and zero in most of space. Both of these issues are solved by smoothing the function γ (Figure 6). The notion of the boundary between the inside and outsides of S has become a little bit more blurry, but the gradient $\nabla\gamma(x)$ is still mostly zero inside and outside S, and is nonzero (but not infinite) in a narrow (but not infinitely narrow) region that contains the surface Ω.

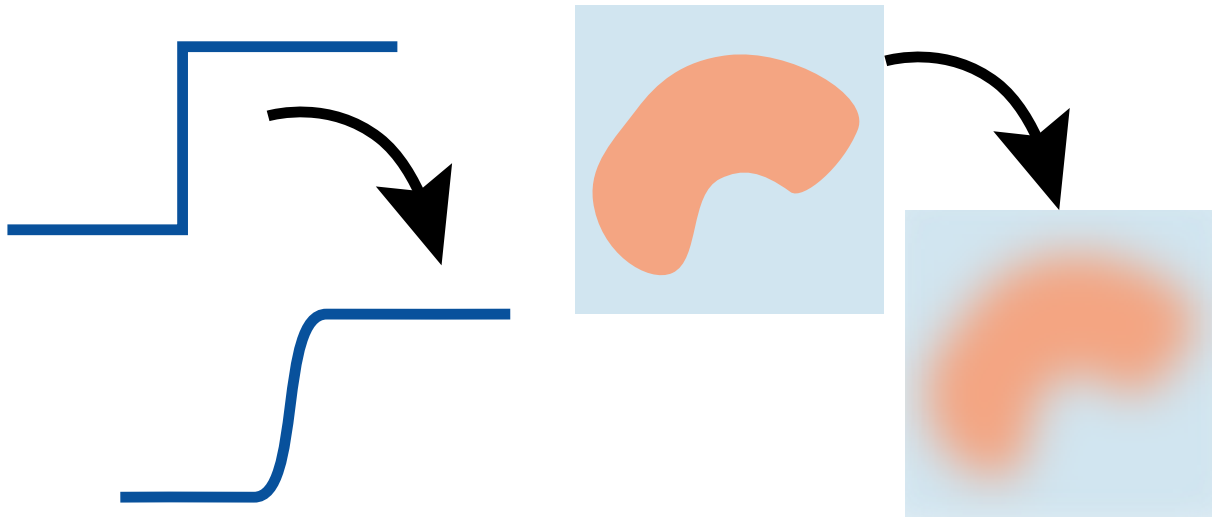

Figure 6: Smoothing a one-dimensional function *(left)* and a two-dimensional function *(right)*.

The problem of blurred indicator function reconstruction can now be formulated as in Definition 3:

Definition 3: Indicator function reconstruction problem

The indicator function reconstruction problem for a point cloud P with normals N is as follows: Find a blurred indicator function γ such that $\nabla\gamma(x_i) = \boldsymbol{n}_i$ for all $x_i \in \mathrm{P}$, $\boldsymbol{n}_i \in \mathrm{N}$.

The problem from Definition 3 can be written as an optimization problem, where we look for the argument function γ that minimizes some energy (an argmin problem):

$$\underset{\gamma}{\operatorname{argmin}} \sum_{i \in \mathrm{P}} \|\nabla\gamma(x_i) - \boldsymbol{n}_i\|^2 + \sum_{j \notin \mathrm{P}} \|\nabla\gamma(x_j)\|^2. \tag{3}$$

(3) is the entirety of the point cloud to indicator function step of PSR. It returns a function γ that is zero on almost all points in space, and whose gradient is equal to the normals at the point cloud. Of course, (3) involves a lot of hand-waving. First, what are the points $j \notin \mathrm{P}$? How do we account for the blurring? And how do we even optimize over the set of all functions γ?

The answers to these problems depend on the particular implementation of PSR. In its very simplest form, the function γ is discretized using a grid (Figure 7). A grid divides all of space into cells of equal volume, and assigns a function value to each vertex of the grid. This is a simple way of turning the uncountable set of all functions in all of space into a simple finite set of degrees of freedom that uniformly cover space.

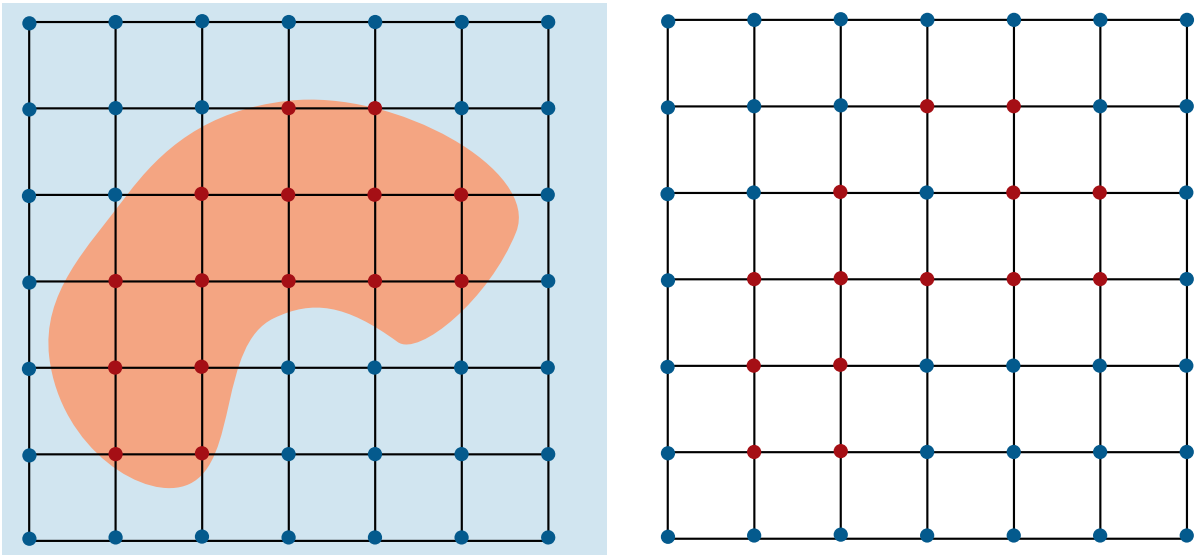

Figure 7: A grid superimposed *(left)* over an indicator function, and on its own *(right)*. A grid is a simple way of discretizing volumetric information.

Using the finite difference scheme (which you will learn about in the exercise accompanying this chapter), we can implement the gradient operator ∇ on a function γ discretized on a grid as a simple matrix: $\nabla\gamma \sim G\gamma$. (3) then becomes:

$$\underset{\gamma}{\operatorname{argmin}} \left\| G \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \cdots \\ \gamma_n \\ \gamma_{n+1} \\ \cdots \\ \gamma_m \end{pmatrix} - \begin{pmatrix} \boldsymbol{n}_1 \\ \boldsymbol{n}_2 \\ \cdots \\ \boldsymbol{n}_n \\ 0 \\ \cdots \\ 0 \end{pmatrix} \right\|^2, \tag{4}$$

where the first n rows represent the inside points (where the normal is provided), and the rows $n+1$ to m represent the outside points (where the normal should be zero).

(4) can easily be solved by inverting a linear system:

$$\begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \cdots \\ \gamma_n \\ \gamma_{n+1} \\ \cdots \\ \gamma_m \end{pmatrix} = \left(G^{\mathrm{T}}G\right)^{-1} G \begin{pmatrix} \boldsymbol{n}_1 \\ \boldsymbol{n}_2 \\ \cdots \\ \boldsymbol{n}_n \\ 0 \\ \cdots \\ 0 \end{pmatrix} \tag{5}$$

After solving (4), we have converted our point cloud into a discrete indicator function on a grid.

Indicator function to triangle mesh

It remains to convert the indicator function γ on the discrete grid to a triangle mesh. To achieve this, we can use the marching cubes (or, for curves in 2D, marching squares) method [5]. This method is conceptually simple, but deviously tricky to implement. Here is how it works for polylines in 2D: Since we have a grid with indicator functions values, and we know that the indicator function is $\gamma(x) = 1$ on the inside of the surface and $\gamma(x) = 0$, we look for each grid cell that contains vertices p_i such that $\gamma(p_i) > \frac{1}{2}$ for some vertices in the cell, and $\gamma(p_j) \leq \frac{1}{2}$ for other vertices in the cell. Then we know that the surface has to pass through that cell, with the and $\gamma(p_j) \leq \frac{1}{2}$ on the outside, and the $\gamma(p_i) > \frac{1}{2}$ points on the inside. The rest is a matter of drawing the line segments correctly such that the right p_i, p_j are on the inside and outside, and so that the segments from different grid cells link up correctly. Figure 8 contains a simple overview of the concept.

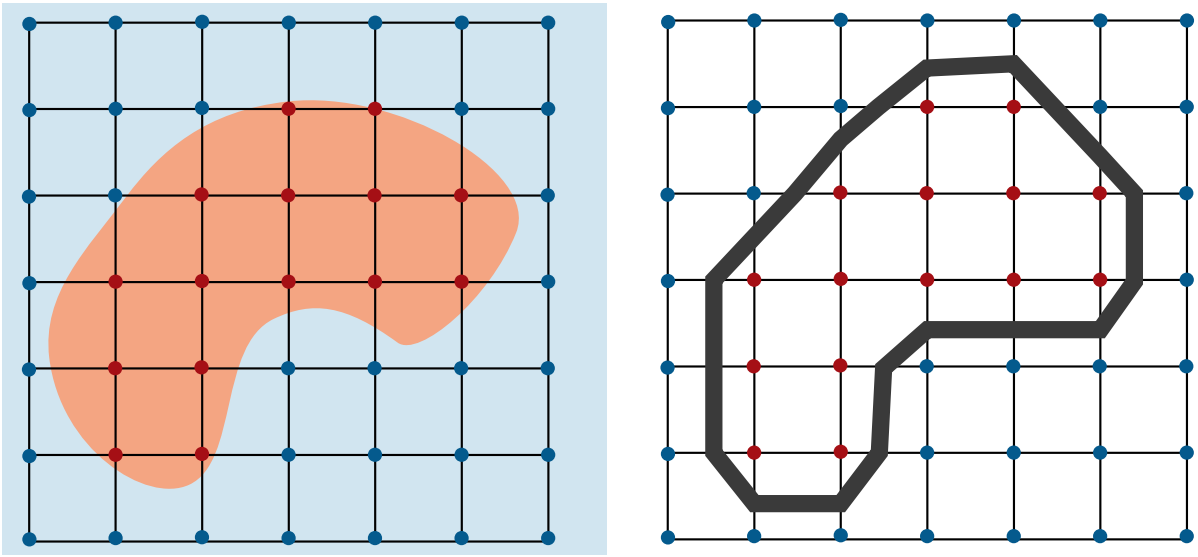

Figure 8: Reconstructing an indicator function *(left)* with the marching squares algorithm *(right)*: Every cell that features both vertices larger as well as smaller than $\frac{1}{2}$ gets a line segment.

This same concept applies to triangle meshes, where it is yet another order of magnitude more difficult to implement: One identifies the 3D cells that contain vertices p_i such that $\gamma(p_i) > \frac{1}{2}$ for some vertices in the cell, and $\gamma(p_j) \leq \frac{1}{2}$, and then fits the right triangular surface patch on each identified cell.

Other point cloud reconstruction algorithms

We have covered PSR, probably the most popular point cloud surface reconstruction algorithm in geometry processing. Science, however, has not stood still since 2006! Many new reconstruction algorithms have come out since, some based on Kazhdan et al.'s PSR approach, and some taking completely different approaches. If you are interested in learning more modern reconstruction methods, I recommend you read some of these articles [6,2,1].

Bibliography

[1] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2Mesh: A Self-Prior for Deformable Meshes. *ACM Trans. Graph.* 39, 4 (2020).

[2] Michael Kazhdan and Hugues Hoppe. 2013. Screened Poisson Surface Reconstruction. *ACM Trans. Graph.* 32, 3 (2013).

[3] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Symposium on Geometry Processing*, 2006.

[4] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. 2000. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*, 2000. 131–144.

[5] William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*, 1987. 163.

[6] Andrei Sharf, Thomas Lewiner, Ariel Shamir, Leif Kobbelt, and Daniel Cohen–Or. 2006. Competing Fronts for Coarse–to–Fine Surface Reconstruction. *Computer Graphics Forum* 25, 3 (2006), 389–398.