An Energy Minimization Approach to Surface Reconstruction

Scott Daniel Gabriel Taubin
Brown University

***Abstract*— In this paper we describe a novel approach to reconstructing a three dimensional digital model from surface samples of a real object. Our approach computes an surface modeled by the sample points from a implicit function defined over a uniform grid. The implicit function is obtained from an energy minimization problem by constraining the gradient field of the function to match the sample normals as closely as possible. Our problem can be discretized into a symmetric positive definite linear system whose solution is the global minimum of the quadratic form. We then present results of our approach and possible methods for improvement.**

***Index Terms*— Surface Reconstruction, Energy Minimization, Conjugate Gradients.**

I. Introduction

In recent years the problem of surface reconstruction has gained a noticable amount of attention from the computer graphics community. Reconstructing digital models from sample data of real world objects had been explored by researchers in previous decades but a motivation to solve the problem efficiently for large models started with the Michelangelo Project.

Over the course of the Michelangelo Project scientists and researchers from Stanford and the University of Washington reconstructed 3D models from Michaelangelo's sculptures in Rome, the largest being Michelangelo's statue of David [7]. Since then other scientific disciplines along with the entertainment and manufacturing industries have relied on surface reconstruction methods to digitize the shape of real objects.

The goal of this paper is to provide an overview of the process of surface reconstruction in the context of our approach. Over the course of this paper we will explain the motivation and mechanics behind surface reconstruction, how we formulate and discretize our approach to the surface reconstruction problem, and how the resulting model can be obtained from the solution to a linear system of equations. Finally, we conclude with the results from our approach along with possible areas for improvement.

II. Related Work

Surface reconstruction is a challenging problem that involves research in efficient algorithms and data structures for generating smooth, detailed models from noisy, irregular, and even missing data. According to [5] the various approaches to the surface reconstruction problem can be grouped into three categories, namely computational geometry techniques, fitting of surface geometry directly to sample points, and fitting an implicit function over the domain of the sample points. In the last method a digital model is generated from an isosurface of the implicit function using an isosurface extraction algorithm such as Marching Cubes.

The focus of this paper will be on the implicit function approach, the goal being to find an implicit function with an isosurface that closesly approximates the surface represented by the point samples. For our purposes an isosurface is a surface, S, in the implicit function, f, where the points in Cartesian space tangent to S all evaluate to the same value [1]. Typically, the isosurface value is zero. All other points in the spatial domain either evaluate to a value greater than zero if they are on one side, such as the exterior, of S and less than or equal to zero if they are on the other side, in this case the interior. One of the benefits of the implicit function approach is that it is not constrained by the topology of the sampled real world object and can thus handle a wide range of real world objects [5].

In a related approach, [9] use an octree that subdivides the space of sample points into local domains . For each leaf in the octree they fit a local quatric function that approximates the sample points in the cell. The global implicit function is then obtained by a proper weighting of the local functions so that they blend across the leaves of the octree. One of the benefits of their approach is that the resulting surface can be ray traced due to the mathematical properties of the local functions.

Another surface reconstruction method described in [5] uses an approach similar to ours by computing an indicator function that is defined as one on the interior of the isosurface and zero elsewhere. They obtain the

indicator function by first computing its Fourier coefficients from surface integrals using Stoke's Theorem. Then they apply an inverse Fourier transform to extract the indicator function from the coefficient values.

Our approach is similar to [9] and [5] in that we also define an implicit function over a spatial grid domain. One significant difference, however, is that we do not need to fit local function approximations nor perform any signal analysis operations.

III. Approach

The general approach we use to solve the surface reconstruction problem is similar to [11]. Rather than attempt to fit an implicit function directly to the sample points we instead place constraints on the implicit function and its gradient. Given a cloud of sample data points and their associated normals obtained from surface M we want to compute an implicit function $f : R^3 \rightarrow R$ such that

$$\forall (p_i, n_i) \in \mathcal{D} \quad f(p_i) = 0 \text{ and } \nabla f(p_i) = n_i.$$

The resulting isosurface is then $M' = \{p | f(p) = 0\}$. The values $f(p)$ are computed as a trilinear interpolation of the function values of f at the lattice points on the uniform grid, thus

$$f(p_i) = \sum_{\alpha \in [p_i]} \phi_\alpha(p_i) f_\alpha$$

where $[p_i]$ denotes the neighborhood of latice points at p_i and ϕ_α is the trilinear weight of the lattice node α. To find the desired isosurface M' we must also compute an appropriate isovalue. The isovalue is found by minimizing the least squares problem $\sum_i (f(p_i) - s)^2$. The solution is the mean of f evaluated at the points in p. Once we have f and s we can use an isosurface extraction algorithm such as Marching Cubes to obtain the resulting polygonal mesh, S, that approximates the surface of M.

Given the above requirements our main objective is to compute an implicit function f whose gradient closely matches the normals of the sample points p. The approach we take is to convert this constraint into an energy minimization problem and find the implicit function f that minimizes a scalar energy function $E(f)$. The first term of the energy function is then

$$\sum_i \mu_i ||\nabla f(p_i) - n_i||^2.$$

We will call this energy term the data contribution. The parameter μ_i is used to weight the contribution of p_i to compensate for non-uniform sampling density. However the single constraint that the gradient of f approximate the sampled normals as closely as possible is not sufficient for solving for an implicit function f.

One of the observations about our current energy function $E(f)$ is that it only constrains the gradient of the implicit function f near the input sample points. This constraint does not dictate what the gradient of f should be further away from the sample data. When we solve for the implicit function we want to ensure that regions far from the data do not evaluate to zero, otherwise they would be considered part of the isosurface M' and would appear as artifacts.

To prevent sudden changes in f that might cause artifacts to appear away from the data points we can impose an additional constraint on the gradient that it must be smooth across the domain of the grid. In smooth functions changes between neighboring points are small. Analogously, we would like to constrain the gradient of the function at a given point in the grid to be close to the gradient of other points in its proximity. This constraint implies that difference in f along the edges of the lattice points should be small. We add this additional constraint to our energy function as

$$\sum_{(\alpha,\beta)} \lambda_{\alpha,\beta} ||\nabla f_\alpha - \nabla f_\beta||^2$$

where α and β are the adjacent lattice nodes of a grid edge and $\lambda_{\alpha,\beta}$ controls how smooth the gradient should be across an edge. We call this additional constraint the regularization term. The energy equation now constrains the gradient of the implicit function over the entire grid domain.

To complete our energy equation we make one last observation. Our last observation comes from the fact that a minimizing function f plus a constant term is also a minimizer of $E(f)$ because the gradient remains unchanged. This observation presents a problem for our current approach because we would like to find a unique function f that minimizes the energy function. In order to constrain f to be the smallest function that minimizes $E(f)$ we add one last energy term

$$\sum_i |f(p_i)|^2.$$

We call this last energy term the data minimizer. With this term we are restricting the minimizing function f to have values as small as possible near the sample points.

The result is that there is now a unique function f that minimizes $E(f)$. In general this last term in will make our linear system symmetric positive definite. We will later see that this is an important attribute when we solve for f numerically. Putting together all the energy terms the problem now becomes one of finding a scalar function f that minimizes the energy function

$$\begin{aligned} E(f) = \quad & \textstyle\sum_i \mu_i ||\nabla f(p_i) - n_i||^2 \\ & + \textstyle\sum_{(\alpha,\beta)} \lambda_{\alpha,\beta} ||\nabla f_\alpha - \nabla f_\beta||^2 \\ & + \textstyle\sum_i |f(p_i)|^2. \end{aligned}$$

IV. Discretization

Now that we have defined constraints for an implicit function that produces M' we focus our attention on how we will discretize $E(f)$ so we can later solve for f numerically. As is common in many partial differential problems we want to discretize the energy function into a system of linear equations using finite differences. In particular, we use a form of the finite difference method known as central differences. Consider the two dimensional case for a function defined over a uniform square grid. The equation for the horizontal gradient component at lattice node α is

$$\nabla f_{\alpha,0} = \begin{cases} f_{(\alpha_0+1,\alpha_1)} - f_{(\alpha_0,\alpha_1)} & \text{if } \alpha_0 = 0 \\ \frac{1}{2}(f_{(\alpha_0+1,\alpha_1)} - f_{(\alpha_0-1,\alpha_1)}) & \text{if } 0 < \alpha_0 < n_0 \\ f_{(\alpha_0,\alpha_1)} - f_{(\alpha_0-1,\alpha_1)} & \text{if } \alpha_0 = n_0 \end{cases}$$

The vertical component is similarly defined by incrementing over α_1 instead of α_0. Since ∇f_α is a homogeneous linear combination of the function values we can rewrite it as

$$\nabla f_\alpha = \sum_\beta \Big[\frac{\partial \nabla f_\alpha}{\partial f_\beta}\Big] f_\beta \tag{1}$$

where $\frac{\partial \nabla f_\alpha}{\partial f_\beta}$ is a constant vector defined according to the positions of α and β. For simplicity we denote this expression as the constant vector $g_{\alpha,\beta}$. Note that $g_{\alpha,\beta}$ is zero if $\alpha = \beta$ or if β is not adjacent to α. Since $f(p)$ is a trilinear combination of f at the nodes $[p]$ we can compute $\nabla f(p)$ as

$$\nabla f(p) = \sum_{\alpha \in [p]} \phi_\alpha(p) \nabla f_\alpha \tag{2}$$

Combining 1 and 2 we can then express $\frac{\partial \nabla f(p)}{\partial f_\beta}$ as

$$\frac{\partial \nabla f(p)}{\partial f_\beta} = \sum_{\alpha \in [p]} \phi_\alpha(p) g_{\alpha,\beta}$$

At this point it would also be helpful to describe the implicit function f as a vector of function values evaluated at each lattice node of the grid. We use a lexicographic ordering such that a node's index into the vector function f is increasing fastest along the x axis, then the y axis, and then the z axis. With these definitions we are ready to discretize our energy function into a linear system of equations.

To begin our discretization process we note that since our energy function is homogenous in the unkown function f we can express it in a quadratic form. A quadratic form is a function of a vector expressed as

$$E(f) = \frac{1}{2} f^T A f - b^T f + c$$

or in the coordinates:

$$E(f) = \frac{1}{2} \sum_\alpha \sum_\beta f_\alpha A_{\alpha\beta} f_\beta - \sum_\alpha b_\alpha^T f_\alpha + c$$

where A is an nxn matrix, b an n length vector, and c a constant. By using matrix calculus we can obtain the entries of the matrix A and vector b from the coordinate form by differentiating with respect to the variables of the function f. Since the algebraic and quadratic form of the energy function are equivalent we conclude that differentiating the algebraic form is the same as computing the entries in the quadratic form. We use γ and μ to designate indices into the variables of f by which we will differentiate, in the case of the matrix A we will always differentiate by f_γ before f_μ. Note also that in the coordinate form the variable γ will correspond to the row of a matrix or vector and μ to the column of a matrix.

From the data contribution we have

$$\begin{aligned} ||\nabla f(p_i) - n_i||^2 &= (\nabla f(p_i) - n_i)^T (\nabla f(p_i) - n_i) \\ &= \nabla f(p_i) \nabla f(p_i) - 2 \nabla f(p_i) n_i + n_i n_i \end{aligned}$$

Then by differentiating

$$\begin{aligned} \frac{1}{2} \frac{\partial}{\partial f_\gamma} \{||\nabla f(p_i) - n_i||^2\} &= \frac{\nabla f(p_i)}{\partial f_\gamma} \nabla f(p_i) - \frac{\nabla f(p_i)}{\partial f_\gamma} n_i \\ \frac{1}{2} \frac{\partial^2}{\partial f_\gamma \partial f_\mu} \{||\nabla f(p_i) - n_i||^2\} &= \frac{\nabla f(p_i)}{\partial f_\gamma} \frac{\nabla f(p_i)}{\partial f_\mu} \end{aligned}$$

From the regularizer term

$$\begin{aligned} &\frac{1}{2} \frac{\partial^2}{\partial f_\gamma \partial f_\mu} \{||\nabla f_\alpha - \nabla f_\beta||^2\} = \\ &(\frac{\nabla f_\alpha}{\partial f_\gamma} - \frac{\nabla f_\beta}{\partial f_\gamma})^T (\frac{\nabla f_\alpha}{\partial f_\mu} - \frac{\nabla f_\beta}{\partial f_\mu}) \end{aligned}$$

And finally from the minimizer term

$$\frac{1}{2}\frac{\partial^2}{\partial f_\gamma \partial f_\mu}\{||f(p_i)||^2\} = \frac{f(p_i)}{\partial f_\gamma}\frac{f(p_i)}{\partial f_\mu}$$

From the definitions above it follows that

$$\begin{aligned} A_{\gamma\mu} &= \frac{1}{2}\frac{\partial^2}{\partial f_\gamma \partial f_\mu}\{E(f)\} \\ &= \textstyle\sum_i \mu_i \sum_{\alpha\in[p_i]} \sum_{\beta\in[p_i]} \phi_\alpha(p_i)\phi_\beta(p_i)\left[g_{\alpha\gamma}^T g_{\beta\mu}\right] \\ &\quad + \textstyle\sum_{(\alpha,\beta)} \lambda_{\alpha,\beta}(g_{\alpha\gamma} - g_{\beta\gamma})^T(g_{\alpha\mu} - g_{\beta\mu}) \\ &\quad + \textstyle\sum_i \mu_i \phi_\gamma(p_i)\phi_\mu(p_i) \\ b_\gamma &= \sum_i \mu_i n_i \sum_{\alpha\in[p_i]} \phi_\alpha(p_i) g_{\alpha\gamma} \end{aligned}$$

V. Energy Minimizer

At this point we now have a quadratic form of our energy equation, $E(f)$, with a known A matrix and b vector. All that is left is to solve for the function f defined over our uniform grid that minimizes $E(f)$. We noted earlier that the matrix A has the property of being symmetric positive definite. A matrix is symmetric positive definite if for every nonzero x, $x^T Ax > 0$. Using this definition and some rules from basic calculus we can derive an equation for finding an implicit function that minimizes the energy equation.

In calculus we can find the critical points of a function by settings its derivative to zero and solving for the unknowns in the resulting equation. In our quadratic form for the energy equation we take its derivative as

$$\begin{aligned} E'(f) &= \frac{1}{2}A^T f + \frac{1}{2}Af - b \\ &= Af - b \end{aligned}$$

The second equality follows from the fact that A is symmetric. Hence, by setting this equation to zero we obtain a linear system $Ax = b$ and by finding a solution to this linear system we have found a critical point of our energy function. We also know that the solution $x = A^{-1}b$ is a global minimum because A is positive definite. To show that the solution x is the global minimum of the energy equation consider another vector y in R^n. From the quadratic form we can show that if A is symmetric then:

$$E(y) = E(f) + \frac{1}{2}(y-f)^T A(y-f)$$

If A is positive definite then by the definition of a symmetric positive definite matrix the second term is positive, thus the energy of E is greater at y than f and f is the global minimum of E.

VI. Linear System Solver

The final step is to obtain the implicit function that minimizes our energy equation by solving a linear system. The most common and straightforward approaches for solving linear systems is Gaussian elimination. However, it requires $O(n^3)$ operations where n is the the dimension of the matrix. It is not atypical for grids to have over 100 nodes along each dimension, corresponding to linear systems having on the order of a million variables. Solving such a system using Gaussian elimination is not very practical.

Another approach to solving linear systems first decomposes the matrix into lower and upper triangular matrices which can then be solved through simple forward and backward substitutions. Due to the fact that our matrix A is symmetric positive definite we can apply a variant of this matrix decomposition approach known as the Cholesky decomposition. Cholesky decomposition decomposes the matrix A into the product of a lower triangular matrix L and it's transpose L^T. Although this approach is potentially more effective than a straightforward Gaussian elimination it suffers from overhead needed to order rows and columns of the matrix in order to reduce fill in [12].

One of the problems that these two methods for solving linear systems do not address is the fact that our matrix A is sparse. Generally speaking, a sparse matrix is a matrix that is dominated by zero entries. This is true of our matrix because for a given lattice node, γ, only a small neighborhood of variables around it will contribute to the row A_γ, in our case only 135 at most. All of the other entries in the row of γ are zero because variable outside its neighborhood do not influence it.

Another branch of linear system solvers that work well with sparse matrices are iterative solvers. Unlike direct solvers like Gaussian elimination or Cholesky decomposition they do not attempt to solve the system directly but instead generate a sequence of approximations that ideally converge to the correct solution. One of the benefits of an iterative approach is that it takes advantage of the fact that most of the entries in a sparse matrix are zero, thereby conserving computational resources only to values that contribute to the result.

To enable efficient linear operations such as vector dot products and matrix-vector multiplications we store our A matrix in a compact column storage (ccs) representation (See Figure 1). In this representation n is the dimension of the matrix, rowndx is the list of nonzero entries per row for each column in the matrix, data is the list of nonzero values corresponding to entries in rowndx, and colptr is a list of indices where each index

```
struct CCS_Matrix {
{
  int     n;
  int*    colptr;
  int*    rowind;
  double* data;
};
```

Fig. 1. C-like psuedo code for a sparse matrix

points to the first entry in rowndx of its column. For example, the first index in colptr always points to the beginning of rowndx since its first entry belongs to the first column of a symmetric positive definite matrix and the next index in colptr will point to the first entry in rowndx that belongs to the second column. Note that since the matrix is symmetric we only store the lower half of the matrix.

Once we've decided on using an iterative solver we have choice of which iterative solver to use. One of the simplest is the Jacobi method, also known as the method of simultaneous displacements. The basic idea behind the Jacobi method is that we start with an initial guess for the implicit function and then solve for each variable independently. After each iteration we update our approximation with the new computed values. The Jacobi method can be written mathematically as

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\Big)$$

One of the benefits of the Jacobi method is that it is easy to implement. However, the Jacobi method is not very effective for large linear systems because it takes several iterations to converge and in some cases may not even converge to the correct solution.

An improved iterative solver is the conjugate gradients method. Although the conjugate gradients method is more complicated than the Jacobi method it is guaranteed to converge in n steps, although it is often the case that fewer steps are required to obtain a good approximation. One of the requirements for using the conjugate gradients method is that it also requires the matrix A to be symmetric positive definite. Further details of conjugate gradients and its derivation are beyond the scope of this paper. A good introduction to the subject can be found in [10].

Although the conjugate gradients method is an improved iterative solver it too can often take a cumbersome number of iterations to converge for large systems. To improve the convergence rate of the conjugate gradients method we can use a matrix preconditioner. A matrix preconditioner is another matrix whose product with A has a lower condition number than A by itself. Generally speaking, linear systems with matrices of a lower condition number converge faster than those with matrices having larger condition numbers. Although multiple preconditioners are available we stick with the simplest one, namely the diagonal preconditioner. The diagonal preconditioner is formed by inverting the diagonal matrix whose entries are the diagonal entries in A.

VII. Results

The results in Figure 2 show the output of our algorithm. The two examples shown are a Chiquita-like bust with over 700K oriented points and an angel with 24K oriented points. These results were produced with 110^3 uniform grids. The faceted looks are partly attributed to per face shading. The surfaces would look smoother with a better shading algorithm like Gouraud shading. Note that the holes in the models are filled. Both the reconstructions were done with standard implementations of the Marching Cubes algorithm.

VIII. Improvements

Although our approach produces decent results it is encumbered by the fact that the quality of the mesh dictates the size of the grid. Our global approach of solving for the implicit function over the entire grid is not scalable. For example, a high quality mesh that captures the fine details of a well sampled model requires on the order of a thousand variables along each grid axis, resulting in over a billion variables. Grids of this order require more memory than available on commodity computers.

In order to compensate for the memory requirement we devised a block based version of our approach. The motivation behind this approach is that we can partition the grid into a collection of disjoint, fixed size blocks and solve for each block independently. This block based approach theoretically enables us to use a large global grid while only needing to store a small subset of it in memory at any given time. The independent nature of the blocks also allows them to be solved in parallel over a distributed computing system such as a cluster.

Although we implemented this approach in the 2D case for relatively small grids we conjecture that this approach will not be amenable to the 3D case with larger grids. One of the problems of the block based approach is that we end up wasting computational resources on blocks with little or no sample points in them. More generally, the problem that this alternative solution does

not address is that more function variables are needed near the surface of M and fewer variables in regions further from the surface.

The idea that we would like to have a varying density of function variables leads us to consider an adaptive approach. In an adaptive approach we would use a hierarchical grid structure such as an octree to hold more variables near the sample data where the surface of M resides and fewer variables further away. By using this adaptive structure for the domain of our implicit function we could apply a hierarchical solver such as the adaptive multigrid method. Other works related to this project show promising results for adaptive methods and it is likely the correct way that we should approach the surface reconstruction problem.

IX. Conclusion

We have presented a novel approach to the surface reconstruction problem whose solution comes from solving the linear system of a symmetric positive definite matrix. Our approach has a straightforward formulation and compact representation. In addition, we are able to apply many of the common linear system solvers that are available, the most effective being the conjugate gradients method with a preconditioner. Although our results are fair we find that there is room for improvement. In future iterations of our work we will use an adaptive approach which we feel will make our results comparable to those obtained in other published research.

References

[1] Bloomenthal J., Wyvill B. 1997. Introduction to implicit surfaces. Morgan Kaufmann Publishers Inc., San Francisco, California.

[2] Bolitho M., Kazhdan M., Burns R., Hoppe H. 2007. Multilevel streaming for out-of-core surface reconstruction. Proceedings of Eurographics symposium on Geometry Processing 2007.

[3] Bolz J., Farmer I., Grinspun E., Schröoder P. 2003. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. ACM SIGGRAPH 2003 Papers.

[4] Briggs, W., Henson, V. E., McCormick, S. 2000. A multigrid tutorial, second edition. SIAM, Philadelphia, Pennsylvania.

[5] Kazhdan, Michael. 2005. Reconstruction of solid models from oriented point sets. Proceedings of Eurographics symposium on Geometry Processing 2005.

[6] Kazhdan, M., Bolitho, M., Hoppe, H. 2006. Poissson surface reconstruction. Proceedings of Eurographics symposium on Geometry Processing 2006.

[7] Levoy, M., Pulli, K., Curless, B., et al. 2000. The digital michelangelo project: 3D scanning of large statues. Proceedings of ACM SIGGRAPH 1999, 131-144.

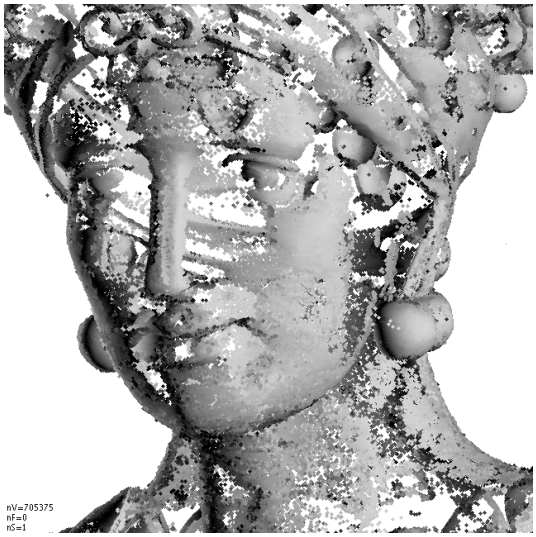
[8] Lorensen W., Cline, H. 1987. Marching cubes: a high resolution 3d surface reconstruction algorithm. Proceedings of ACM SIGGRAPH 1987, 163-169.

[9] Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. 2003. Multi-level partition of unity implicits. ACM Trans. Graph. 22, 3, 463-470.

[10] Shewchuck, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps`

[11] Sibley, P., Taubin, G. 2005. Vectorfield isosurface-based reconstruction from oriented points. ACM SIGGRAPH 2005 Sketch.

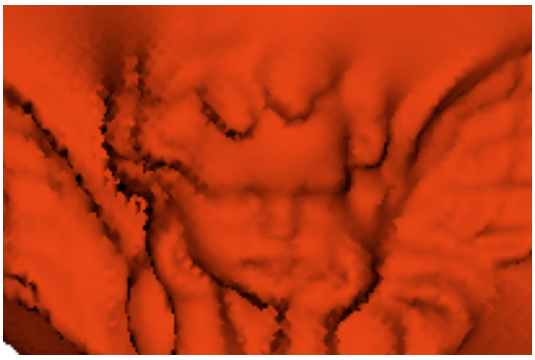
[12] Stewart, G. W. 2003. Building an old-fashioned sparse solver. `http://www.cs.umd.edu/~stewart`



(a) Chiquita Points

(b) Chiquita Reconstruct

(c) Angel Points



(d) Angel Reconstruct

Fig. 2. Reconstruction results