

Tracing implicit surfaces from polygonal mesh using FIGTree

HONORS PAPER BY KHOA HA
ADVICE BY RAMANI DURAISWAMI

University of Maryland, College Park
khoaha@umd.edu

Abstract

This paper examines brute-force, Gaussian, and FIGTree methods to create implicit surface approximations of polygonal mesh. For the latter methods iterative procedures are used to solve vertex weights, allowing for more form-fitting traces. All the tests are then run against differently-sized test sets and their performances are compared. The merits of the FIGTree approach and its applications are examined.

I. OVERVIEW

In “Interpolating and Approximating Implicit Surfaces from Polygon Soup” [1], the authors discuss using implicit surface modeling to repair problematic polygon meshes by approximating them as a summation of Gaussian functions (known as a “blobby model”). However, they cite the operation as prohibitive due to the high runtime cost of computing the implicit function. By using the FIGTree method to estimate the implicit mesh instead of exact Gauss with least squares regression the running complexity can be reduced from $O(MN)$ to $O(M + N)$, where M is the number of basis functions and N is the number of evaluation points.

There is much practical value in a tool to efficiently correct polygonal mesh. By converting to implicit mesh and back, problems such as discontinuity (mismatched edges, holes, etc) and duplicate features can be corrected in one calculation. By varying the precision, level of detail models can also be generated easily, which is useful for applications such as computer games. Additionally there are times when an implicit surface is more desirable than its polygonal counterpart, such as when performing collision testing. Thus, this process has applications both in modeling and real-time environments. It can also be useful for fur-

ther computation such as solving differential equations since it allows the polygonal mesh can be regenerated to have evenly-sized faces, etc.

This paper will discuss the application of FIGTree in tracing polygons and test the results against a naïve “brute force” approach using k -nearest neighbor search and a non-optimized blobby model construction using Gaussian functions.

II. GENERAL ALGORITHM

This algorithm will process polygonal meshes represented as a set of n vertices and their corresponding faces. These values are scaled to lie within the unit cube. We compute the normals as the cross product of two adjacent edges on a face. We want to find the implicit function

$$f(x_i, y_i, z_i) = 0, \quad i = 1, \dots, n \quad (1)$$

where (x_i, y_i, z_i) is a single vertex. To do this we discretize the unit cube into a point matrix according to a specified resolution, with each point assigned a value relating its distance to the mesh. The values for each cell can be determined according to a brute-force approach, a more versatile Gaussian-based method, or a FIGTree approximation of the Gaussian method. Afterwards the native iso-surface or marching cubes algorithm can de-

termine the implicit equation from the grid of values.

III. BRUTE-FORCE APPROACH

To determine the value at each cell, k -nearest neighbor search is used to find the Euclidean distance to the nearest vertex on the mesh which serves as an estimate for the shortest distance to the mesh surface. To find the implicit equation, we want the isosurface such that the distance to the mesh is zero. With the discrete grid, we can determine that the figure lies between the positive points (points outside the mesh) and the negative points (points inside the mesh). In order to do this the distance estimates must be signed. For this reason we compute the angle between the vector to the closest vertex and the normal at the vertex. The value is signed as positive if the difference is greater than 90 degrees, and signed as negative otherwise. From this either the marching cubes or built-in isosurface method can be used to extract the surface.

BRUTE-FORCE ALGORITHM

```

for point  $p \in \text{meshgrid}$  do
   $\text{index}, \text{error} \leftarrow \text{knnsearch}(\text{vertices}, p)$ 
   $p' \leftarrow \text{vertex}[\text{index}]$ 
   $\text{vec} \leftarrow p - p'$ 
   $\text{angle} \leftarrow \text{atan}(\text{norm}(\text{normal}[\text{index}] \times$ 
 $p'), \text{normal}[\text{index}] \cdot p')$ 
   $\text{output}_p \leftarrow \text{sign}(\text{angle} - \pi) \cdot \text{error}$ 
end for

```

EFFICIENCY

For N vertices enclosed in a three-dimensional space composed of M cells, this algorithm requires $O(M \log N)$ time to populate the grid. This is because for each point in the space the nearest vertex must be found, and the vertices are stored in a k - d tree by default for the search call. The isosurface tracing portion of the algorithm will require $O(M^3)$ time, however this is present in all forms of this algorithm and is

not dependent on the vertex complexity N of the polygonal mesh.

IV. PRECONDITIONING WEIGHTS

Instead of having each point in space be influenced by the nearest vertex, we can construct an interpolation using n blobby functions. However, when summing the influence of mesh vertices, often times we find some vertices should have more impact than others. This is because the distribution of vertices on the model surface may not be uniform and closely clustered vertices will exert more influence on the resulting implicit mesh than isolated vertices (see Figure 1.) To address this, we attempt to solve the interpolation problem by identifying fixed points where the isovalue must conform and employ a generalized minimum residual method to estimate weights for each vertex. [2]

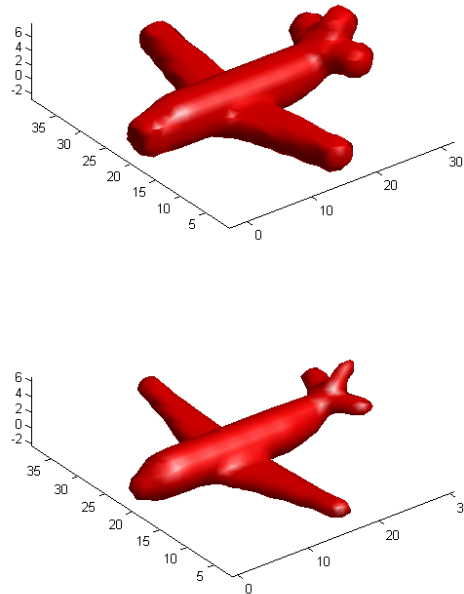


Figure 1: (Above) Plane model with no vertex weighting, (Below) plane model with weights. Note the more tapered features on the weighted figure in areas with more vertex detail.

IMPLEMENTATION

To find the fixed points, we take each vertex, find its corresponding “off-point” by traversing a certain amount δ along its normal, and fixing that to be a certain value

$$f(x_i + \delta n_{x_i}, y_i + \delta n_{y_i}, z_i + \delta n_{z_i}) = \alpha \quad (2)$$

(we will use $\alpha = 1$ in this case). This provides a rough isosurface sample to match against. The generalized minimum residual method is an iterative procedure which solves the equation

$$Ax = b, \quad A_{ij} = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(p_i - p_j)^2}{2\sigma^2}} \quad (3)$$

for x , our weights. A is a linear transform, supplied by the method of our choosing. [3] Each term in the matrix is a Gaussian function evaluated between two points in our set, p_i and p_j . The bandwidth, σ is provided by the user and will vary based on the model. In our tests this was typically found through trial-and-error. The matrix is provided as a closure function with all parameters including the vertices and off-points supplied. Solving the linear system with Gauss requires $O(M^3)$ operations due to the required back-substitution. FIGTree requires $O(M)$ operations by employing iterative approximations, avoiding back-substitution. [4][5] b is a matrix of the values corresponding each off-point. The method initially guesses the zero solution and iteratively refines the estimate until a desired epsilon is reached or the maximum number of iterations has occurred (which was the case in our trials). These weight estimates are then fed back into the function when evaluating the spatial matrix for the actual values (see Figure 2 for a comparison of the original model with normals and its trace.)

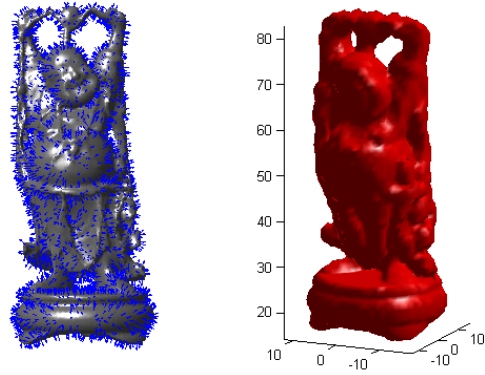


Figure 2: (Left) Original Buddha model with normals highlighted, (Right) FIGTree trace.

V. GAUSSIAN IMPLEMENTATION

The Gaussian summation function takes a list of vertices, a list of evaluation points, a list of weights per vertex, and a bandwidth. For each point it finds the distance to each vertex, calculates the influence of that particular vertex by evaluating a one-dimensional Gaussian with the specified bandwidth at the value corresponding to the distance between the point and the vertex. These influences are then multiplied by their corresponding weights and summed per point. This summation function is used once for the generalized minimum residual method to compute weights and then run again at each spatial point to recover the actual iso-values.

GAUSSIAN ALGORITHM

```

for index = 1  $\rightarrow$  n do
  offpoints[index]  $\leftarrow$  vertex[index] +  $\delta$   $\cdot$ 
  normal[index]
  Let f(weights) = gaussian(vertex,
  offpoints, weights, bandwidth)
  weights  $\leftarrow$  gmres(f,  $\alpha$ )
end for
output  $\leftarrow$  gaussian(vertex, offpoints,
weights, bandwidth)
  
```

EFFICIENCY

The efficiency of Gaussian summation is $O(MN)$ since for every point in the space, the Gaussian at each vertex must be evaluated and summed. There is an additional $O(M^3)$ term if weighting is applied due to solving the interpolation problem. This is slower than the brute-force approach but allows for more fine-tuned controls for each vertex's contribution and will more often produce smoother results.

VI. FIGTREE IMPLEMENTATION

First the generalized minimum residual method is run using FIGTree to compute weight estimates for each vertex. Then the vertex list, weights, and array of spatial points is passed to the FIGTree algorithm, which selects an epsilon-exact estimate for the sum of influences at each point.

FIGTREE ALGORITHM

```

for  $index = 1 \rightarrow n$  do
   $offpoints[index] \leftarrow vertex[index] + \delta \cdot normal[index]$ 
  Let  $f(weights) = FIGTree(vertex, bandwidth, weights, offpoints, error)$ 
   $weights \leftarrow gmres(f, \alpha)$ 
end for
 $output \leftarrow FIGTree(vertex, bandwidth, weights, offpoints, error)$ 

```

EFFICIENCY

Finding the potentials in the three-dimensional grid requires $O(M + N)$ time, an improvement over the previous $O(M \log N)$ time under the brute-force approach and the $O(MN)$ time required by the Gaussian method.

ACCURACY

When applying the FIGTree algorithm, there are four primary factors which affect the accuracy of the resulting isosurface. The first is the subdivision of the three-dimensional grid. A higher resolution would lead to more accurate

results since more samples would be computed. A lower resolution might be desirable though for creating simplified meshes for applications such as level-of-detail-models (see Figure 3.) The second is the mesh complexity. More vertices correspond to more sources for Gaussians which lead to more fine-grained results. Third is the kernel bandwidth for the Gaussian function. If the bandwidth is too high, the surface would be loosely-fitted. If the bandwidth is too low, the surface would have dips between vertices. Lastly is the error cutoff. A higher tolerance would lead to less accurate estimates at each point and allow for looser surfaces, while a lower one would not.

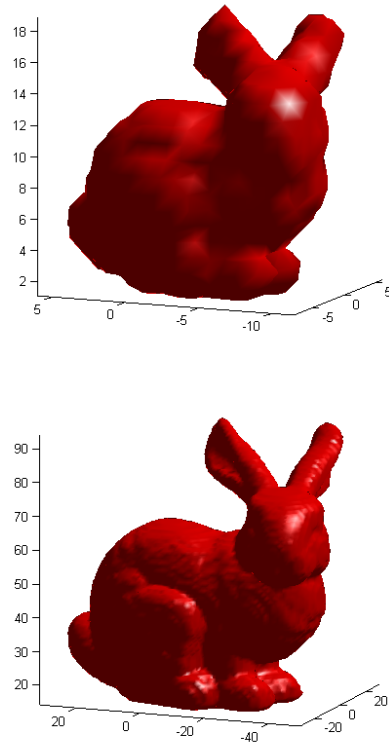


Figure 3: (Above) Stanford bunny traced via FIGTree at a resolution of 20, (Below) same model traced via FIGTree at a resolution of 100.

VII. RESULTS

To compare the three implicit mesh algorithms we sample a number of points on the unit sphere and run the each algorithm on the set. For the Gaussian and FIGTree methods the bandwidth and normal offset were kept fixed but the isovalue was allowed to vary per set since the output of the weight estimation could vary. All tests were run at a unit resolution of 10 (see Table 1.)

The brute-force approach has the crudest reconstruction, owing to using the nearest vertex as an estimate and applying no smoothing function. This is especially apparent in the smaller sample sets, which exhibit large bumps and even tearing. The Gaussian method produced much better reconstructions, likely due to the bandwidth smoothing the results. The residual for the weight estimations is quite large though, with a residual of 200 on the 500 set. This indicates a large error at the end of the iteration, however the spherical distribution of the data set hides this fact. On the 500 set the Gaussian method took over 400 seconds, which renders the method largely impractical, even compared to the brute-force method. The Gaussian method was not run on the 5000 set due to the substantial run time required. The FIGTree method yields good reconstructions at all levels, with weight residuals below 1/100 for each. The time to compute each set is also much quicker than the other methods, with the time for the 5000 set beating the times for both brute-force and Gauss on the 50 set.

VIII. CONCLUSION

On our test set, the FIGTree method ran the fastest at all levels and also had the lowest

residual error for weighting, indicating quick convergence. Additionally, since the iteration number for the generalized minimum residual method and FIGTree error can be specified for each use, the algorithm can be made arbitrarily precise such that it produces no additional error over Gauss. Because of these factors it makes an excellent candidate for tracing implicit mesh. Additional results from traces can be seen below (see Table 2.)

IX. FUTURE WORK

A problem encountered while testing the FIGTree method was that the parameters for bandwidth, grid resolution, and error margins needed to be chosen through trial-and-error for the best results. It would be better if these parameters were automatically selected based on the user's needs. One solution could be to provide a toolbox where the user could specify the level of detail to preserve and noise tolerance, and have an algorithm select the parameters from there. This would lessen the work needed to acquire ideal results and greatly extend the usefulness of the method. Other specific optimizations that could be made to the algorithm include applying it to geometry captured by 3D scanners, in which the vertex density is relatively consistent, and to generating level-of-detail models in which we would want to preserve prominent details while also simplifying extraneous geometry. While the FIGTree method won't be fast enough for real-time use in most cases, it still greatly outspeeds the other methods and can be used for pre-computation in most use cases where implicit surface tracing is common today.

A. COMPARISONS

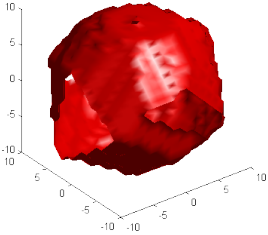
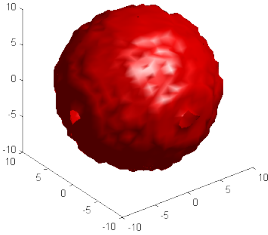
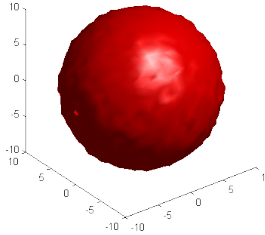
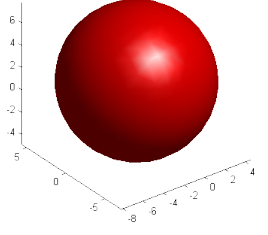
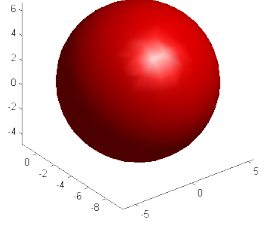
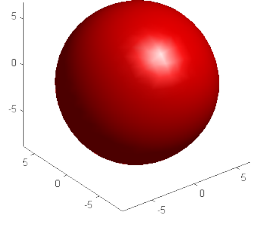
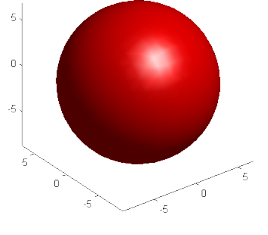
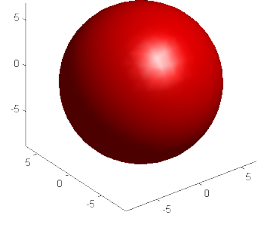
Method	50 points	500 points	5000 points
Brute-force			
Gaussian			(No image)
FIGTree			

Table 1: Visual comparison of each method based on random sampling of a sphere.

B. MEASUREMENTS

The following are the parameters used to produce the results for each image. The running time and generalized minimum residual are also included (if applicable). For FIGTree, the allowable error threshold is listed as well. All tests were performed on an Intel Core-i5 machine with default RAM allocations. Brute-force and Gaussian tests were run in 64-bit MATLAB 2012b, while FIGTree tests were run in 32-bit MATLAB 2010a for compatibility reasons.

Image	Time (s)	Resolution	Normal offset	Bandwidth	Threshold	Isovalue	Residual error
Plane w/ weights	0.266152	40	0.001	0.025	0.0001	0.5	0.025
Buddha statue	3.950326	100	0.001	0.005	0.0001	0.025	0.0068
Low-res bunny	15.141674	20	0.005	0.02	0.0001	0.05	0.012
High-res bunny	3.864089	100	0.005	0.005	0.0001	0.025	0.087
Brute w/ 50	12.892367	10	N/A	N/A	N/A	0	N/A
Brute w/ 500	25.974247	10	N/A	N/A	N/A	0	N/A
Brute w/ 5000	108.063739	10	N/A	N/A	N/A	0	N/A
Gauss w/ 50	24.329785	10	0.01	1	N/A	19.5	18
Gauss w/ 500	462.369360	10	0.01	1	N/A	199.417	200
FIGTree w/ 50	0.024721	10	0.01	1	0.0001	1.2	0.00026
FIGTree w/ 500	0.236726	10	0.01	1	0.0001	1.2	0.00099
FIGTree w/ 5000	1.175478	10	0.01	1	0.0001	1.2	0.00018

Table 2: All parameters for presented images

C. CODE

To execute the code, first run either `genSphere.m` or `loadModel.m` to construct the geometry. The `toolbox_graph` library is used to read `.ply` files and must be on the same path for `loadModel.m` to work. [6] Then run `runBrute.m`, `runGauss.m`, or `runTree.m` to generate output for the brute-force method, Gaussian method, or FIGTree method, respectively.

GAUSSIAN.M

```
1 function output = gaussian(vertex,list,weights,bandwidth)
2     output = zeros(size(list, 1), 1);
3     for h = 1 : size(list, 1)
4         for j = 1 : size(vertex, 1)
5             output(h) = output(h) + normpdf(0, pdist([list(h,:);
6                 vertex(j,:)])*weights(j), bandwidth);
7         end
8     end
9 end
```

GENSPHERE.M

```
1 % generate point cloud
2 n = INPUT_NUMBER_HERE; % number of sample points
3 vertex = zeros(n,3);
4 for i = 1 : n
5     phi = 2*pi*rand(1);
6     theta = 2*pi*rand(1);
7     vertex(i,1) = sin(phi)*cos(theta);
8     vertex(i,2) = sin(phi)*sin(theta);
9     vertex(i,3) = cos(phi);
10 end
11 normal = vertex;
12 res = INPUT_NUMBER_HERE; % set the desired resolution
```

LOADMODEL.M

```
1 % load model
2 name = 'airplane.ply';
3 options.name = name;
4 [vertex,faces] = read_ply(name);
5 vertex = vertex/SCALING_FACTOR; % correct model to lie within unit
6     cube
7 [normal,normalf] = compute_normal(vertex,faces);
8 options.normal = normal;
9 % render
10 clf; plot_mesh(vertex,faces,options); shading interp; axis tight;
11 options.normal = [];
12
13 normal = normal.';
14 res = INPUT_NUMBER_HERE; % set the desired resolution
```



```
15  isovalue = INPUT_NUMBER_HERE; % the value at which to trace the
    surface

RUNBRUTE.M

1  % create meshgrid
2  [x,y,z] = meshgrid(-res:res,-res:res,-res:res);
3  data = zeros(size(x,1),size(x,2),size(x,3));
4
5  tic;
6
7  % compute nearest point for data
8  for i = 1 : size(x,1)
9      for j = 1 : size(x,2)
10         for k = 1 : size(x,3)
11             a = [x(i,j,k),y(i,j,k),z(i,j,k)]/res;
12             [idp,r] = knnsearch(vertex,a);
13             b = a-vertex(idp,:);
14             % sign with the normal
15             angle = atan2(norm(cross(normal(idp,:),b)),dot(normal(
16                 idp,:),b));
17             data(i,j,k) = sign(angle-1.5708)*r*res;
18         end
19     end
20 end
21
22 toc;
23
24 % render
25 s = isosurface(x,y,z,data,isovalue);
26 p = patch(s);
27 isonormals(x,y,z,data,p);
28 set(p,'FaceColor','red','EdgeColor','none');
29 view(3); daspect([1 1 1]); axis tight;
30 camlight;
    lighting gouraud;
```

RUNGAUSS.M

```
1  offset = INPUT_NUMBER_HERE; % displacement amount along normal
2  bandwidth = INPUT_NUMBER_HERE; % gaussian parameter
3
4  % create meshgrid
5  [x,y,z] = meshgrid(-res:res,-res:res,-res:res);
6  data = zeros(size(x,1),size(y,2),size(z,3));
```

```
7
8 % build eval points
9 r = 2*res+1;
10 list = zeros(r^3,3);
11 for i = 1 : size(x,1)
12     for j = 1 : size(y,2)
13         for k = 1 : size(z,3)
14             list((i-1)*r^2+(j-1)*r+k,1) = (i-res)/res;
15             list((i-1)*r^2+(j-1)*r+k,2) = (j-res)/res;
16             list((i-1)*r^2+(j-1)*r+k,3) = (k-res)/res;
17         end
18     end
19 end
20
21 tic;
22
23 % apply pcg for weights
24 offpts = zeros(size(vertex,1),3);
25 fixed = ones(size(vertex,1),1);
26 for i = 1 : size(vertex,1)
27     offpts(i,:) = vertex(i,:) + offset*normal(i,:);
28 end
29 weights = gmres(@(p) gaussian(vertex,offpts,p,bandwidth), fixed);
30
31 % sum gaussians
32 output = gaussian(vertex, list, weights, bandwidth);
33
34 for i = 1 : size(x,1)
35     for j = 1 : size(y,2)
36         for k = 1 : size(z,3)
37             data(i,j,k) = output((i-1)*r^2+(j-1)*r+k);
38         end
39     end
40 end
41
42 toc;
43
44 % render
45 s = isosurface(x,y,z,data,isovalue);
46 p = patch(s);
47 isonormals(x,y,z,data,p);
48 set(p,'FaceColor','red','EdgeColor','none');
49 view(3); daspect([1 1 1]); axis tight;
50 camlight;
51 lighting gouraud;
```

RUNTREE.M

```
1 offset = INPUT_NUMBER_HERE; % displacement amount along normal
2 bandwidth = INPUT_NUMBER_HERE; % gaussian parameter
3 error = INPUT_NUMBER_HERE; % precision level for FIGTree
4
5 % create meshgrid
6 [x,y,z] = meshgrid(-res:res,-res:res,-res:res);
7 data = zeros(size(x,1),size(y,2),size(z,3));
8
9 % build eval points
10 r = 2*res+1;
11 list = zeros(3,r^3);
12 for i = 1 : size(x,1)
13     for j = 1 : size(y,2)
14         for k = 1 : size(z,3)
15             list(1,(i-1)*r^2+(j-1)*r+k) = (i-res)/res;
16             list(2,(i-1)*r^2+(j-1)*r+k) = (j-res)/res;
17             list(3,(i-1)*r^2+(j-1)*r+k) = (k-res)/res;
18         end
19     end
20 end
21
22 tic;
23
24 % apply pcg for weights
25 offpts = zeros(size(vertex,1),3);
26 fixed = ones(size(vertex,1),1);
27 for i = 1 : size(vertex,1)
28     offpts(i,:) = vertex(i,:) + offset*normal(i,:);
29 end
30 weights = gmres(@(p) figtree(vertex.',bandwidth,p,offpts.',error),
31     fixed);
32
33 % run figtree
34 output = figtree(vertex.',bandwidth,weights,list,error);
35 for i = 1 : size(x,1)
36     for j = 1 : size(y,2)
37         for k = 1 : size(z,3)
38             data(i,j,k) = output((i-1)*r^2+(j-1)*r+k);
39         end
40     end
41 end
42 toc;
43
```

```
44 % render
45 s = isosurface(x,y,z,data,isovalue);
46 p = patch(s);
47 isonormals(x,y,z,data,p);
48 set(p,'FaceColor','red','EdgeColor','none');
49 view(3); daspect([1 1 1]); axis tight;
50 camlight;
51 lighting gouraud;
```

REFERENCES

- [1] C. Shen, J. F. O'Brien, and J. R. Shewchuk, "Interpolating and approximating implicit surfaces from polygon soup," Computer Graphics Proceedings and Annual Conference Series, 2004.
- [2] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, "Reconstruction and representation of 3d objects with radial basis function," SIGGRAPH, 2001.
- [3] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," SIAM Journal on Scientific and Statistical Computing, vol. 7, pp. 856–869, 1986.
- [4] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, and L. S. Davis, "Automatic online tuning for fast gaussian summation," NIPS, vol. 21, pp. 1113–1120, 2008.
- [5] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov, "Fast computation of sums of gaussians in high dimensions," 2005.
- [6] G. Peyre, "Toolbox graph," <http://www.mathworks.com/matlabcentral/fileexchange/5355-toolbox-graph>, 2004–2009.