Variational Volumetric Surface Reconstruction from Unorganized Points

ID: paper1039

Abstract

Reconstruction of smooth surfaces from point sets is an important problem in many applications since this sort of data often appear in real-life scenarios. This paper presents method for solving this this problem at interactive rates by means of second order energy minimization obtained by solving $\triangle^2 = 0$ on a discrete scalar field using an iterative multigrid approach.

Every step in the reconstruction process takes place on the GPU. Consequently, the surface is immediately available for any standard volume visualization method utilizing packed volume textures. This eliminates the need for lare amounts of texture transfers or a costly polygonization step.

1. Introduction

The need for interpolation of data values associated with scattered points is something that arises in a vast number of scenarios and a wide range of dimensions. A well studied problem in computer graphics and computer vision is the reconstruction of surfaces from point samples, and one approach to this problem is through the use of scattered data interpolation. The basic idea is to find a function which has a *level set* that either interpolates or approximates the input points. There are many advantages to this strategy for surface reconstruction: A wide range of techniques for converting level set surfaces to triangle meshes are available, and numerous techniques for scattered data interpolation can be brought to bear on the problem. For instance, radial basis function techniques have recently become popular for that purpose [TO99] [SPOK95].

In the work presented here, we take the approach of producing a level set representation, but the embedding scalar field is discretely represented in a voxel grid. This is not in itself novel. Instead our contribution is to find the scalar field which minimizes certain energy functionals: In particular the bending energy. This turns out to lead to surfaces which are fair yet interpolate the surface.

Moreover, the method is amenable to a multigrid implementation (See Section 3.3.). Consequently, it is very efficient, requiring relatively few iterations at the most detailed levels, and in many cases, we can reconstruct a volumetric surface representation on a 256^3 grid in less than a second for relatively noise free data using our GPU based implementation.

Our method is a component in a larger framework of streaming kernels for volume processing implemented on the GPU which is discussed in Section 5. A particular virtue of our method in this context is that all processing (after the initial points have been transferred to the GPU memory) takes place on the GPU, and there is no need to transfer the volume back to the CPU since further processing and rendering takes place entirely on the GPU.

Finally, it is possible to control the degree of confidence in the point samples. A confidence of 1 forces the reconstructed surface through the point while a smaller confidence indicates that approximation is acceptable. For laser scanned point sets, it is usually desirable to set the confidence to less than 1 since the data set is likely to contain noise. In Section 6 we discuss these trade offs, provide timings and compare to other methods both in terms of quality and performance.

2. Related Work

There is a great deal of literature on the reconstruction of surfaces from points. Arguably, these methods can be divided into three categories.

- Methods which produce implicit surfaces.
- Methods which form surfaces by connecting the points – often based on an initial Delaunay tetrahedralizaion of the point set. A good example is the well known power crust algorithm due to Amenta et al. [ACK01].

- Methods which fit an existing surface representation to the point cloud. At the same time such methods often strive for a fair and compact representation. A good example is the work by Hoppe et al. [HDD*93].

Below, we will restrict the discussion to methods which reconstruct implicit representations.

One of the first surface reconstruction methods which made few assumptions about the topology of the object and the nature of the point cloud was due to Hoppe et al. [HDD*92]. The basic idea is to reconstruct locally a linear signed distance field for each point. The global distance field is then pieced together from these local distance fields, and the surface is built by contouring. Curless et al. also created a (pseudo) distance field, but used more information about the scanning modality [CL96], and Davis et al. introduced hole filling by diffusion [DMGL02]. This is reminiscent of our method, but they simply used straight averaging to propagate voxel information, hence no energy minimization took place, and the method was only used locally to fill holes.

Finding a set of basis functions which forms an implicit surface reconstruction of a point set, is a tricky problem which has attracted a great deal of attention. One of the early examples is Muraki's method which tried to infer a metaball model from a range scan [Mur91]. Unfortunately, the method was fairly slow since both metaball position and other parameters had to be inferred. More recently, *radial basis functions* have been introduced to the graphics community by [TO99] and, earlier, in a different formulation by Savchenko et al. [SPOK95]. A linear combination of radial basis functions is found such that the 0 level set interpolates or approximates the input points. The combination is found by solving a large linear system. One problem with RBF based methods is the need to place points inside and/or outside the object in order to have non-zero points for the function to interpolate. Another issue is the fact that the linear system is both large and dense. The fast multipole method improves performance [CBC*01], and the system can be made sparse if compactly supported basis functions are used [MYC*01]. Ohtake et al. considered an incremental approach using multi-scale RBFs [OBS03]. Shen et al. incorporated normal information in the basis functions and interpolated triangle primitives rather than points [SOS04]. Although great improvements have been made, radial basis functions remain relatively costly, and although the final evaluation can be made on the graphics processor, the entire process maps poorly to the GPU. The same is true of the more efficient multilevel partition of unity method due to Ohtake et al. [OBA*03].

Some authors have used the *level set method* to reconstruct surfaces from point data [ZOF01, Whi98]. This involves defining a speed function which attracts the level set surface to the data points. The level set method is highly flexible and it is possible to include confidence measures and smoothing terms in the speed function. However, it is also a fairly complex machinery, and frequent reinitializations are typically needed to keep the representation close to a distance field.

In [Kaz05] Kazhdan demonstrates that using Stoke's theorem it is possible to reconstruct the Fourier transform of the indicator function (which is 0 outside and 1 inside a solid) from a set of point samples.

In recent work [KBH06], Kazhdan et al. propose an equivalent method for surface reconstruction based on solving a Poisson problem on an adaptive grid. Specifically, a slightly blurred version of the indicator function is reconstructed. The blurring is necessary because the indicator function is clearly not differentiable, and the problem boils down to finding a function whose gradient field matches the (smoothly interpolated) vector field induced by the point samples. From the reconstructed indicator function, a surface is extracted by contouring.

Our inspiration for this work came in part from methods for smoothing triangulated manifolds. It is well known [KCVS98] that the membrane and bending energies of a function f are minimized by solving $\Delta f = 0$ and $\Delta^2 f = 0$, respectively. If f is a surface represented by a triangle mesh, and we define a Laplace operator for triangle meshes, we can minimize the membrane and bending energies by solving the mentioned equations.

Much of the speed of the method is due to the GPU-based multigrid implementation. In [GWL*03] there is a discussion of the issues involved in implementing a multigrid solver using programmable graphics hardware. However, note that they are mostly concerned with 2D Poisson problems.

3. Reconstruction

We consider the volume to represent a scalar field $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, and the aim of our algorithm is to minimize an energy $E[f]$ subject to the constraint that $f(\vec{p}_i) = v_i$ for a set of *interpolation conditions*. These conditions are defined by the point set. The energy is either the membrane energy

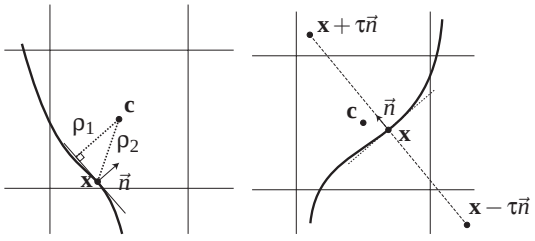
$$E_M[f(\mathbf{x})] = \int f_x^2 + f_y^2 + f_z^2 \qquad (1)$$

which is minimized by $\Delta f = 0$ or the bending energy

$$E_B[f(\mathbf{x})] = \frac{1}{2} \int f_{xx}^2 + f_{yy}^2 + f_{zz}^2 + 2f_{xy}^2 + 2f_{xz}^2 + 2f_{yz}^2 \ . \qquad (2)$$

which is minimized by $\Delta^2 f = 0$. As we shall see, minimizing the former energy is somewhat faster, but interpolation is more reliably achieved using the latter. All computations are performed numerically by discretizing the Laplace operator. The discretized version, $\mathcal{L}$, is defined in appendix A.

The interpolation conditions, $\mathbf{p}_i$ are placed at voxel centers: From the plane defined by an input point and its normal, we create three constraints in the vicinity of the input point.



(a) surface samples (b) normal samples

Figure 1: *Calculating of the interpolating conditions for a voxel centered at* **c** *for a sample located at* **x**

The constraint location $\mathbf{p}_i$ is a voxel center, and the voxel value, v_i, is the plane distance. This process is described in detail in Section 3.1.

Subsequently, we use Jacobi iteration in order to solve the PDE, i.e. to minimize $\triangle f$ or $\triangle^2 f$. This process is described in detail in Section 3.2.

Unfortunately, this process tends to converge very slowly, and if too few iterations are used, the desired smoothness is not obtained, nor are large holes closed. For this reason, we start at very low resolution, typically $4 \times 4 \times 4$ where all voxels receive a constraint and then progress to higher resolutions. Each time the resolution is increased, we interpolate the solution at the previous level to the more detailed level. We compute novel voxel constraints and then solve again using Jacobi iteration as before. Typically, we progress until we reach a resolution of $256 \times 256 \times 256$ which is where the graphics hardware limit is encountered. Since we use the solution at a lower level to initialize the solution at a higher level, the solution converges far more quickly than if we simply started at the highest resolution. The details of this multigrid scheme are provided in Section 3.3.

3.1. Generating Interplation Conditions

The compution of the interpolation conditions is done by assigning a value to each voxel enclosing a sample (here we construe a voxel as a small box). The value is calculated as the distance ρ between the voxel center **c** and the surface from which the sample is taken. As illustrated in figure 1(a), two different approximations of ρ are used: ρ_1 which is the the signed distance between **c** and the plane defined by the sample location **x** and normal $\vec{n}$, and ρ_2 which is the euclidian distance between **x** and **c**.

In the case of multiple samples located in the same voxel, the distance value for the particular voxel can be determined by picking the closest based on ρ_2, whereas the actual distance value used when solving the PDE is ρ_1.

Each sample contributes to more than the single voxel located near the surface as described in the previous. From a surface sample located at **x**, two normal samples are defined by

$$\hat{\mathbf{x}} = \mathbf{x} \pm \tau\vec{n} \ , \tag{3}$$

where τ is magnitude of the offset vector defined in both the positive and negative normal direction such that normal samples are introduced both inside and outside the surface as illustrated in figure 1(b). For a particular normal sample, the value of ρ_2 is calculated using the position of the corresponding surface sample **x** in order to ensure that surface samples are not culled away by normal samples. Likewise, the plane equation needed to calculate ρ_1 is the same for both surface and normal samples.

The purpose of the extra samples is the same as the purpose of the normal constraints used for RBF interpolation which is to ensure the gradient of the reconstructed surface function evaluated at the location of a particular point sample is close to the normal of the sample. The magnitude τ of the offset vector is chosen such that it is proportional to the voxel spacing of the discretization grid.

3.2. PDE Solving

The idea behind the algorithm presented here is to formulate minimization of the bending energy for a trivariate function $f(x,y,z)$ represented discretely as a grid of scalar values, d. A voxel in the grid will be denoted d_i where i is the grid index.

Using the squared Laplace operator defined in appendix A , the discrete partial differential equation to solve in order to minimize the bending energy can be written

$$\mathcal{L}^2 d = 0 \ . \tag{4}$$

This PDE is solved using Jacobi iteration with a simple update rule for a particular voxel value d_i.

$$d_i \leftarrow d_i - \frac{\delta}{\nu}\mathcal{L}^2(d_i) \ , \tag{5}$$

where ν is used as a weight for each neighboring voxel. For a particular case where a 6 voxel neighborhood is used, ν becomes $7/6$. The damping factor δ is used for stability. The present experiments use $\delta = 0.4$.

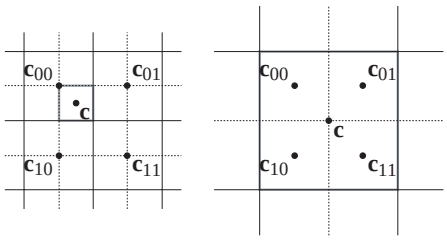
For the voxels used in the present context, there is a distinction between voxels with an interpolation condition and those without. To incorporate this distinction in the model, the confidence value w is used to dampen the smoothing additionally. The complete update rule becomes

$$d_i \leftarrow d_i + (1-w)\frac{\delta}{\nu}\mathcal{L}^2(d_i) \ , \tag{6}$$

where $w = 0$ everywhere except at voxels with an interpolation condition. The confidence value can then be used to force the surface to interpolate the point samples by setting $w = 1$.

3.3. Multigrid Solution

On a fine grid, the convergence of (6) is very slow. To speed up the convergence, a multigrid approach is applied. The operators for resolution changing are illustrated in figure 2, where the coarse-to-fine operator, known as *prolongation* and fine-to-coarse, known as *restriction*, as shown. As seen,



(a) Prolongation (b) Restriction

Figure 2: *Resolution changes for grids where the value at a voxel centered at* **c** *is calculated by interpolating between* $\mathbf{c}_{ij}$*. The destination voxel is highlighted in grey*

both operators are easily defined simply by performing a trilinear interpolated lookup of the destination voxel centered in **c** in the source grid, thus using the voxel values from the source grid centered at $\mathbf{c}_{ij}$ to calculate the value at **c**. The two-dimensional illustrations above, extend trivially to three dimensions.

Since the point samples are independent of grid resolution, the interpolation conditions are recomputed after each resolution change. Hence, the precision of the interpolation conditions gets better with increasing resolution.

We do not impose boundary conditions. Voxel indices are simply clamped to the indices of the boundary voxels. Consequently, the outside neighbour of a boundary voxel is itself.

4. Implementation

We perform the operations on the volume by utilizing the capabilities to render directly to textures available in modern graphics hardware using the concept of *frame buffer objects* defined in OpenGL. This way we implicitly loop over each voxel by means of the rasterization step in the rendering pipeline. The processing of a fragment corresponds to performing the actual calculation of the value for the corresponding voxel. Since most voxel operations use very small kernels, the amount of work done in each fragment is not very large.

4.1. Voxel packing/unpacking

We represent the voxels of the scalar field by packing them into 2D textures. The reason a 3D texture is not used, is that the complete volume cannot be bound to the framebuffer at once.

The packing is done in such a way that the slices of the 3D volume are laid out as tiles in the 2D texture as shown in figure 3.

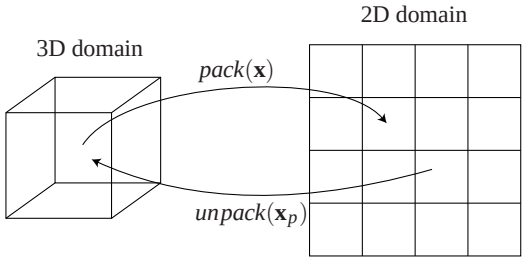


Figure 3: *Overview of voxel packing*

The function $pack(\mathbf{x})$ does the conversion from a voxel cell $\mathbf{x} = [x,y,z]^T$ to a grid cell $\mathbf{x}_p = [x_p, y_p]^T$ and is defined

$$pack([x,y,z]^T) = \begin{bmatrix} x + D_x(z \mod S) \\ y + D_y \lfloor z/S \rfloor \end{bmatrix} , \qquad (7)$$

where D is the resolution of the voxel grid, and S is the number of tiles in each row of the packed 2D texture. The resolution of the packed 2D texture is

$$D_x S \times D_y (D_z/S) \ , \qquad (8)$$

which imposes the constraint that D_z must be divisible by S.

The inverse of $pack(\mathbf{x})$ is the $unpack(\mathbf{x}_p)$ which gives the voxel cell for a given position in the packed texture:

$$unpack([x_p, y_p]^T) = \begin{bmatrix} x_p \mod D_x \\ y_p \mod D_y \\ \lfloor x_p/D_x \rfloor + S \lfloor y_p/D_y \rfloor \end{bmatrix} . \qquad (9)$$

4.2. Updating

Values of the scalar field are modified by the GPU by rendering a viewport-sized quadrilateral to the particular 2D texture. We use two separate textures to employ a ping-pong scheme, where one texture is read-only input and the other is write-only output. Iterative updates are thus performed by alternating the role of each texture. The current values of the scalar field can at any time be found in the texture last written to.

The operations are implemented in fragment programs. We choose a fixed number of iterations on each level. This is a fairly common approach, but we might save some iterations or get better precision if we used a threshold on the residual to select when to stop. On graphics hardware this is only possible if we measure the residual in the L^∞ norm using occlusion queries [GWL*03]. Unfortunately, these queries would introduce considerable latency.

5. Applications and Rendering

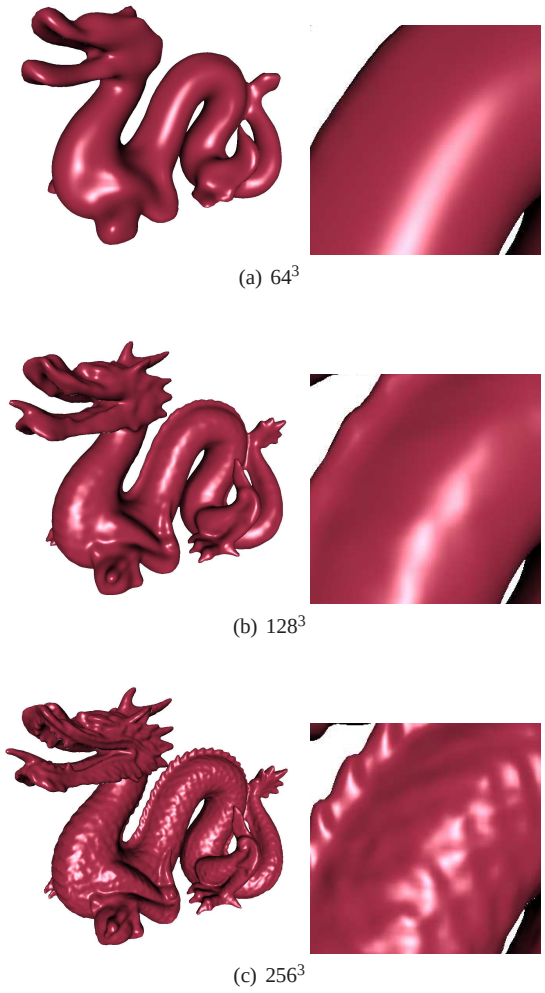
The iterative nature of the method makes it useful for several applications for the reconstructed shape.

- Boolean operations with analytically defined primitives are easy to implement by evaluating the resulting implicit function in every voxel.
- Shapes represented by point sets do not define a proper volume measure. The volume of the shape described by the scalar field can be computed by counting the number of inside voxels using an occlusion query. The accuracy of this measure is of course highly dependant on the resolution of the voxel grid.
- The scalar field can be turned into a distance field by using re-initialization [JBS06] which is easily implemented on the GPU.

To render the shape we ray-cast the volume using fragment programs as described in [KW03]. During ray traversal, the voxels along the ray are unpacked from the packed texture using the *unpack* function implemented within the fragment program. Since our data is an approximate distance field, we can use the value of d to determine the step length along the ray. Because of this very few samples along the ray are needed to find the surface intersection. In the pixels near the silhouette where the ray is nearly tangent to the surface, more samples are needed. However, the dynamic loop capabilities of modern GPUs makes this issue less problematic since silhouette pixels in general only occupy a small percentage of the entire rendered image.

Once the intersection point is found, the surface normal is calculated from the gradient which can be approximated using central differences. With these properties we employ simple Phong-shading, but other more elaborate shading schemes could be used, see [HSS*05].

Since we render directly from the texture on which we perform the voxel operations, we are able to calculate several iterations of a number of manipulation tools between the rendering of each frame while still maintaining real-time frame rates.



(a) 64^3

(b) 128^3

(c) 256^3

Figure 4: *Stanford dragon reconstructed at different resolutions*

6. Results and Discussion

We now describe the results obtained. The system used is equipped with a Nvidia Geforce 8800GTX graphics card with 768 mb video memory.

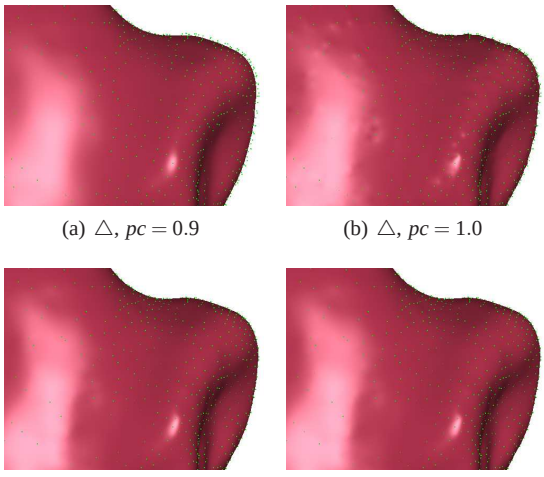
The effect of increasing the resolution is shown in Figure 4.

The two energy minimization operators are applied in Figure 5. The use of w as a way of forcing interpolation is seen in b) and d) whereas approximation is performed in a) and c).

The multigrid approach is mainly used to speed up the convergence of the iterations to the PDE solution. A nice side effect is that holes are closed without compromising the minimization of the bending energy (Figure 6).

As seen in Figure 7, the multigrid approach facilitates the reconstruction of interpolating surfaces and gives a significant speed-up in the reconstruction of approximating surfaces.

The computation times of the shown renderings are listed in table 1, where n_f refer to the number of iterations used at the finest grid and n_c the number on the preceding coarse grids. To test the effect on the reconstruction time of the number samples, we construct the dragon model at different decimation levels. Figure 8 show that when the number of samples is increased by a factor of 500 (from 3536 to 1767812), the construction time is increased by only 43%.



(a) $\triangle$, $pc = 0.9$ (b) $\triangle$, $pc = 1.0$

(c) $\triangle^2$, $pc = 0.9$ (d) $\triangle^2$, $pc = 1.0$

Figure 5: *Ear impression reconstructed using different diffusion operators and regularization parameters. Note that the points are shown.*

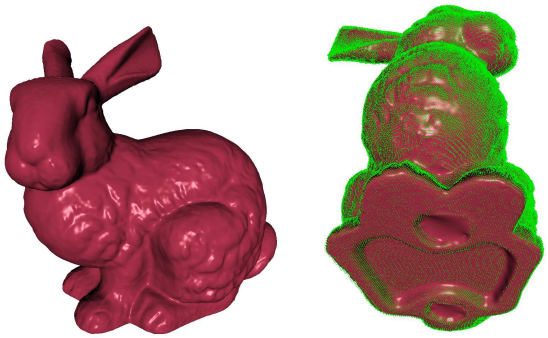
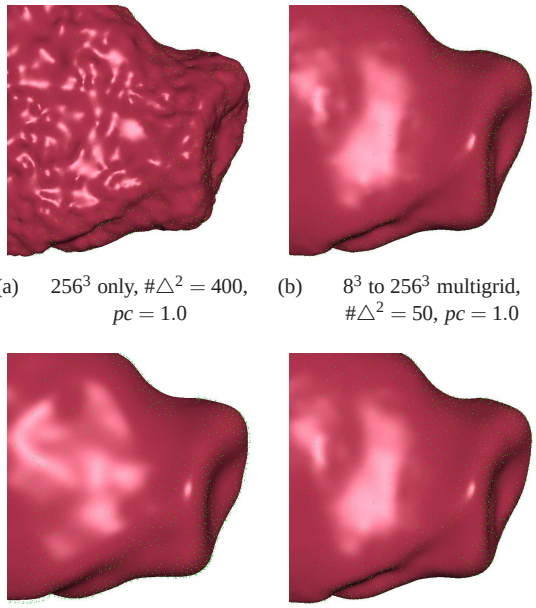


Figure 6: *Reconstruction of the Stanford bunny. Notice how the holes are filled nicely*



(a) 256^3 only, $\#\triangle^2 = 400$, $pc = 1.0$ (b) 8^3 to 256^3 multigrid, $\#\triangle^2 = 50$, $pc = 1.0$

(c) 256^3 only, $\#\triangle^2 = 1000$, $pc = 0.9$ (d) 8^3 to 256^3 multigrid, $\#\triangle^2 = 50$, $pc = 0.9$

Figure 7: *Ear impression reconstruction of interpolating (a,b) and approximating (c,d) surfaces with and without multigrid. In (f), the $\triangle^2$ operator is only applied at the finest level*

time	model	fig.	res.	w	n_c	n_f
2.1	dragon	5(a)	64^3	0.9	100	50
3.1	dragon	5(b)	128^3	0.9	100	50
6.7	dragon	5(c)	256^3	0.9	100	40
0.3	ear ($\triangle$)	6(a,b)	256^3		1	3
1.8	ear ($\triangle^2$)	6(c,d)	256^3		10	20
3.1	bunny	7	256^3	0.9	20	30
4.6	ear (m)	8(b,d)	256^3		0	50
39	ear (s)	8(a)	256^3	1.0		400
91	ear (s)	8(c)	256^3	0.9		1000

Table 1: *Reconstruction times (in seconds) of the shown models. The two bottom rows are single grid timings.*

7. Discussion and Conclusions

Our method is very fast compared to other methods for surface reconstruction for unorganized points. It is hard to make a truly fair comparison between one's own method and those used in other papers. That being said, the time to reconstruct the Stanford Bunny from the original scans is usually on the order of minutes. According to Table 2 in [KBH06], the multilevel partition of unity method is amongst the fastest, reconstructing a mesh model in 28 seconds. We produce a volume in 3.1 seconds as shown in Table 1.

Our method is a part of a larger package of GPU based stream kernels which we use to manipulate and render volume data. Thanks to this framework there is rarely any need to read the volume back from graphics memory, and the only communication overhead is the initial point transmission.

One disadvantage of the method is that the amount of surface detail we can reconstruct is currently limited by the maximum texture size which does not permit us to use volumes larger than $256 \times 256 \times 256$. Compared to the adaptive volume method used by Kazhdan et al. [KBH06] this might seem unimpressive. Fortunately, we seem to be able to obtain slightly better precision at the *same* level of resolution (compare Figure 4(c) to Figure 3 in [KBH06]) in roughly a quarter of the time. One particular reason for this might be the fact that the method by Kazhdan et al. reconstructs a

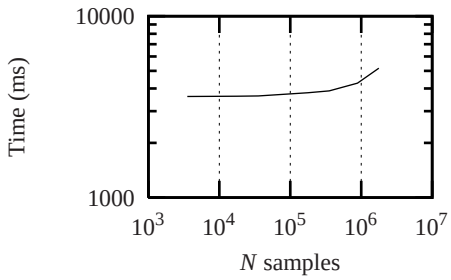


Figure 8: *Relation between point set size and reconstruction time*

slightly blurred version of the indicator function (the indicator function is 1 inside the solid and 0 outside), and, unless the indicator function belongs to a sphere there is no level-set corresponding precisely to the unblurred, true shape. Put differently, a slight smoothing of the shape is an integral part of their scheme.

We provide several "knobs" for configuring the method to particular situations. The user can choose

- Minimizing the membrane or bending energy.
- Number of Jacobi iterations at each level.
- Point confidence.

Minimizing the membrane energy is faster – the operator is simpler and requires fewer iterations. However, if the point confidence is smaller than 1, shrinkage is often pronounced. The slightly slower bending energy is better at coping with situations where we want the surface to interpolate. To aid selection of the right scheme we offer three scenarios:

- In the case of a very dense point cloud, almost every voxel near the surface receives a point splat. In this case, it is fairly unimportant which energy is minimized since the appearance of the surface is largely (even if not entirely) determined by the point splats. The multigrid scheme is still fairly important however, since it ensures that the inside/outside information is efficiently propagated and that holes are closed. The Laplacian scheme is fast and sufficient.
- Most point clouds are noisy to some degree. In this case, a point confidence of 1 is clearly inappropriate since we do not want to model the noise. Either the Laplacian or the square Laplacian could be used. The issue is mostly shrinkage. If the cloud is sparse, shrinkage tends to be exacerbated because there are fewer constraints, hence the square Laplacian scheme is preferable.
- In the case of a point cloud with relatively little noise, we clearly wish to set the point confidence to 1 in order to get precise interpolation. Assuming the cloud is sparse, the results are invariably poor using the (non-square) Laplacian as shown in Figure 5(a). In this case, the square Laplacian scheme must be used.

8. Future Work

Presently, we set the point confidence for all points. This is simple but not the best solution. A better solution would be to use some real confidence measure for each point.

References

[ACK01] AMENTA N., CHOI S., KOLLURI R.: The power crust. *Proceedings of the sixth ACM symposium on Solid modeling and applications* (2001), 249–266.

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 67–76.

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. *Proceedings of SIGGRAPH 1996* (1996), 303–312.

[DMGL02] DAVIS J., MARSCHNER S. R., GARR M., LEVOY M.: Filling holes in complex surfaces using volumetric diffusion. In *First International Symposium on 3D Data Processing Visualization and Transmission (3DPVT'02)* (2002).

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *Proceedings of Graphics Hardware 2003* (San Diego, CA, July 2003).

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *Proceedings of ACM SIGGRAPH 1992* (1992), 71–78.

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. *Proceedings of SIGGRAPH 1993* (1993), 19–26.

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M.: Real-time ray-casting and advanced shading of discrete isosurfaces. In *Proceedings of Eurographics 2005* (2005), pp. 303–312.

[JBS06] JONES M. W., BÆRENTZEN J. A., SRAMEK M.: 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics* (2006).

[Kaz05] KAZHDAN M.: Reconstruction of solid models from oriented point sets. In *Eurographics Symposium on Geometry Processing* (2005), pp. 73–82.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing* (2006), pp. 61–70.

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J.,

SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *ACM SIGGRAPH '98 proceedings* (1998), pp. 105–114.

[KW03] KRÜGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *Proceedings of IEEE Visualization 2003* (2003).

[Mur91] MURAKI S.: Volumetric shape description of range data using "blobby model". *ACM SIGGRAPH Computer Graphics 25*, 4 (1991), 227–235.

[MYC*01] MORSE B. S., YOO T. S., CHEN D. T., RHEINGANS P., SUBRAMANIAN K. R.: Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SMI '01: Proceedings of the International Conference on Shape Modeling and Applications* (2001), IEEE Computer Society.

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.: Multi-level partition of unity implicits. *International Conference on Computer Graphics and Interactive Techniques* (2003), 463–470.

[OBS03] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *Shape Modeling International 2003* (2003), pp. 153–161.

[SOS04] SHEN C., O'BRIEN J. F., SHEWCHUK J. R.: Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004* (Aug. 2004), ACM Press, pp. 896–904.

[SPOK95] SAVCHENKO V., PASKO A., OKUNEV O., KUNII T.: Function Representation of Solids Reconstructed from Scattered Surface Points and Contours. *Computer Graphics Forum 14*, 4 (1995), 181–188.

[TO99] TURK G., O'BRIEN J. F.: Shape transformation using variational implicit functions. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 335–342.

[Whi98] WHITAKER R.: A level-set approach to 3d reconstruction from range data. *International Journal of Computer Vision 29*, 3 (1998), 203–231.

[ZOF01] ZHAO H., OSHER S., FEDKIW R.: Fast surface reconstruction using the level set method. *1st IEEE Workshop on Variational and Level Set Methods, 8th ICCV 80*, 3 (2001), 194–202.

Appendix A: Energy Minimization

For a trivariate function, the bending energy is described by the functional

$$E_B[f(\mathbf{x})] = \frac{1}{2}\int f_{xx}^2 + f_{yy}^2 + f_{zz}^2 + 2f_{xy}^2 + 2f_{xz}^2 + 2f_{yz}^2 \ . \quad (10)$$

This energy is minimized using Euler's equation where the above functional is set to zero. The integrand F is defined by

$$E_B[f(\mathbf{x})] = \frac{1}{2}\int F(f_{xx}, f_{yy}, f_{zz}, f_{xy}, f_{xz}, f_{yz}) \ ,$$

and Euler's equation is used to obtain

$$\begin{aligned}\frac{\partial E}{\partial f} = &\frac{\partial^2}{\partial x^2}\frac{\partial F}{\partial f_{xx}} + \frac{\partial^2}{\partial y^2}\frac{\partial F}{\partial f_{yy}} + \frac{\partial^2}{\partial z^2}\frac{\partial F}{\partial f_{zz}} \\ &+ \frac{\partial^2}{\partial x\partial y}\frac{\partial F}{\partial f_{xy}} + \frac{\partial^2}{\partial x\partial z}\frac{\partial F}{\partial f_{xz}} + \frac{\partial^2}{\partial y\partial z}\frac{\partial F}{\partial f_{yz}} \ .\end{aligned}$$

The first terms expands to

$$\begin{aligned}\frac{\partial^2}{\partial x^2}\frac{\partial F}{\partial f_{xx}} &= \frac{\partial^2}{\partial x^2}\frac{\partial (f_{xx}^2 + \ldots)}{\partial f_{xx}} \\ &= \frac{\partial^2}{\partial x^2}(2f_{xx} + \ldots) \\ &= 2f_{xxxx} \ ,\end{aligned}$$

and similar calculations for the remaining terms lead to the equation

$$\begin{aligned}\frac{\partial E}{\partial f} = &f_{xxxx} + f_{yyyy} + f_{zzzz} \\ &+ 2f_{xxyy} + 2f_{xxzz} + 2f_{yyzz} \ .\end{aligned}$$

This result is simply the squared Laplace operator. To minimize the bending energy, we solve

$$\begin{aligned}\triangle^2 f = &f_{xxxx} + f_{yyyy} + f_{zzzz} \\ &+ 2f_{xxyy} + 2f_{xxzz} + 2f_{yyzz} = 0 \ .\end{aligned} \quad (11)$$

Similar calculations give the ordinary Laplace operator

$$\triangle f = f_{xx} + f_{yy} + f_{zz} = 0 \ , \quad (12)$$

which in terms minimize the *membrane energy* defined in (1).

Equation (11) is solved through iterative updates of the individual voxels, where the value d represents the voxel value. We use the discrete Laplace operator $\mathcal{L}(d)$

$$\mathcal{L}(d) = \frac{1}{n}\sum_{d_i \in N(d)} (d_i - d) \ , \quad (13)$$

where the neighborhood $N(d)$ a 6 voxel "plus" kernel. In other words, the Laplacian $\mathcal{L}(d)$ is defined for a voxel d as the sum of differences between d and voxel values d_i in the neighborhood divided by the number of neighbors.

The squared Laplacian $\mathcal{L}^2(d)$ is simply defined as the Laplacian of the Laplacian of each neighbor.

$$\mathcal{L}^2(d) = \frac{1}{n}\sum_{d_i \in N(d)} (\mathcal{L}(d_i) - \mathcal{L}(d)) \ . \quad (14)$$