LoD Volume Rendering of FEA Data

Shyh-Kuang Ueng* Yan-Jen Su† Chi-Tang Chang‡

Department of Computer Science
National Taiwan Ocean University
No. 2, Pei-Ning Road, Keelung, Taiwan 202

ABSTRACT

In this article, a new multiple resolution volume rendering method for Finite Element Analysis (FEA) data is presented. Our method is composed of three stages: At the first stage, the *Gauss points* of the FEA cells are calculated. The function values, gradients, diffusions, and *influence scopes* of the Gauss points are computed. By representing the Gauss points as graph vertices and connecting adjacent Gauss points with edges, an *adjacency graph* is created. The adjacency graph is used to represent the FEA data in the subsequent computation. At the second stage, a hierarchical structure is established upon the adjacency graph. Any two neighboring vertices with similar function values are merged into a new vertex. The similarity is measured by using a user-defined threshold. Consequently a new adjacency graph is constructed. Then the threshold is increased, and the graph reduction is triggered again to generate another adjacency graph. By repeating the processing, multiple adjacency graphs are computed, and a *Level of Detail*(LoD) representation of the FEA data is established. At the third stage, the LoD structure is rendered by using a *splatting method*. At first, a level of adjacency graph is selected by users. The graph vertices are sorted based on their visibility orders and projected onto the image plane in *back-to-front* order. *Billboards* are used to render the vertices in the projection. The function values, gradients, and influence scopes of the vertices are utilized to decide the colors, opacities, orientations, and shapes of the billboards. The billboards are then modulated with texture maps to generate the footprints of the vertices. Finally, these footprints are composited to produce the volume rendering image.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms;

Keywords: Volume rendering, splatting method, level-of-detail, unstructured data, scientific visualization

1 INTRODUCTION

Finite Element Analysis(FEA) is a powerful technique used in scientific computing [1, 32]. *Volume rendering* is a popular method for visualizing FEA data. By generating semi-transparent cloud images, volume rendering can reveal the blending and distribution of a function inside a 3D domain. Therefore the global behavior of the function is depicted. This functionality is difficult to obtain by using other skills, like cross-section and iso-surface visualization. Volume rendering methods can be categorized into two types, the *ray-casting methods* and the *projection methods*. In a ray-casting method, rays are casted from the eye position through the pixels on the image plane toward the FEA data. For each ray casted, function values are sampled along the ray and converted into colors and opacities. The colors are composited to produce the value of the pixel penetrated by the ray[6, 7, 10]. In the projection methods, the cells, the vertices, or other selected points of the FEA data are projected onto the image plane in *front-to-back* or *back-to-front* order. The effects of the projections are accumulated to form the final image [15, 19, 29]. Some hybrid methods have also been developed to reduce computing costs and to increase image quality [8, 21, 31].

1.1 Related Work

Nonetheless, volume rendering is still a slow process, especially when the data size is large. Many techniques have been developed to speed up the computation. In [8, 21, 13], parallel volume rendering algorithms are proposed to speed up the processing. In [15, 19], the projections of FEA cells are approximated by using polygons. *Scanline algorithms* for polygon filling are employed to compute the values of the pixels covered by the projections. Other researchers design advanced *splatting methods* by using texture maps and graphics hardware to reduce computing time. These algorithms are presented in [5, 17, 18, 28, 33]. Ueng et al. developed an *out-of-core* algorithm for visualizing large vector field data [26]. The FEA data is divided into sub-regions based on an octree structure. Only the sub-regions required in the computing are brought into the main memory. Therefore the required memory capacity is reduced, and large data sets can be interactively processed in a desktop machine. Another out-of-core method is proposed in [31]. The data are compressed in a preprocessing stage. During the rendering process, the data are uncompressed and rendered on the fly by using hybrid volume rendering skills.

Recently many *multiresolution visualization* techniques are proposed for rendering volume data. In these methods, the data are pre-processed to form *level-of-detail*(LoD) representations. In the visualization processing, data from one or several adjacent levels of the LoD structures are rendered to produce the final image. Therefore, the size of data involved in the processing is reduced, and the rendering speed is increased. In [12], Laur and Hanrahan resample volume data based on an octree structure, and hence the data are stored in hierarchical data structures. Lower resolution data are rendered for quick visualization while higher resolution data are rendered for exploring the details. A similar approach is adopted in [16] for visualizing large irregular data. In [2, 4, 3, 9, 20], new algorithms are presented to create LoD representations for tetrahedral meshes. In these methods, the LoD structures are built up by gradually simplifying and approximating the original meshes to support multiple resolution visualization. Another method is designed by LaMar et al. in [11]. The FEA data is encoded based on an octree structure. Unlike other methods, the viewing direction and distance of the eye position are taken into consideration in the rendering processing. The regions closer to the viewer are rendered with higher accuracy while other regions are visualized in lower resolution.

*e-mail: skueng@mail.ntou.edu.tw
†e-mail: m91570009@mail.ntou.edu.tw
‡e-mail:ykq@cyber.cs.ntou.edu.tw

IEEE Visualization 2004
October 10-15, Austin, Texas, USA
0-7803-8788-0/04/$20.00 ©2004 IEEE

1.2 Summary

A new LoD visualization method for FEA data is presented in this paper. At first, the FEA mesh is converted into an *adjacency graph* by generating and connecting the *Gauss points* of the FEA data. Then the adjacency graph vertices are merged to generate multiple adjacency graphs based on user-defined criteria. These adjacency graphs serve as the LoD representation of the FEA data. In order to achieve better rendering quality, each adjacency graph vertex is associated with several attributes. The basic attributes consist of the function value and coordinates of the vertex. Other significant attributes include: the *influence scope* which is a sphere representing the influence extents of the vertex, the gradient which shows the direction in which the function value varies most, and the *diffusion* which estimates the degree of the variation of the function value. In the rendering processing, a level of adjacency graph is selected by users. Then the vertices of the graph are projected onto the image plane in back-to-front order to produce the image. To speed up the computation, the vertices are rendered by using *billboards*. The colors, opacities, shapes, and orientation of the billboards are computed, based on the attributes of the vertices. The billboards are then modulated with texture maps and composited to generate the image. Compared with other methods, our method possesses the following advantages: First, our approach is meshless. No mesh generation, region-refinement, or cell coalescing is required for constructing the LoD structure. Therefore our approach is easier to implement. Furthermore our method can also be extended to cope with scattered data. Second, during the rendering computation, the targets to be rendered are the adjacency graph vertices. The geometrical objects passed into the graphics pipeline are a set of points. Therefore the costs of culling, clipping, and rendering are reduced. Third, once the LoD structure is created, the FEA mesh is no longer needed. Only the adjacency graph vertices are stored. Therefore less memory space is needed for keeping the hierarchical structure.

This paper is organized as follows: In Section 2, the methodology of creating the LoD representation is presented. Then in Section 3, our splatting rendering method is described. The usage of the vertex attributes are explained there. Test results and analysis are shown in Section 4. Conclusion and future work are provided in Section 5.

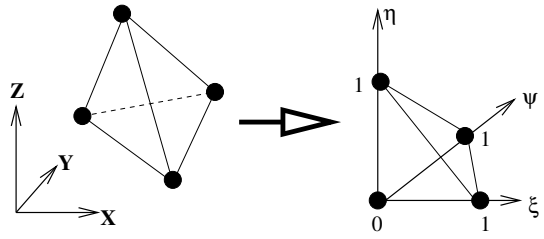
2 Construction of the LoD Representation

The LoD representation of the FEA data consists of multiple layers of adjacency graphs. The bottom layer graph is comprised with the Gauss points of the FEA cells. This graph is gradually reduced to create other layers of adjacency graphs to form the multiple resolution representation.

2.1 Gauss Points and Function Reconstruction

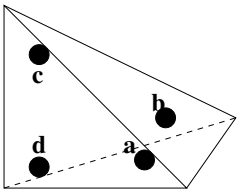
In splatting rendering, projecting an FEA cell can be regarded as carrying out integration inside the cell along the projection direction. *Gauss quadrature method* is a popular technique for calculating integration in FEA computing [32]. To evaluate the integration of a function in an element, the element is transformed into a template cell, the *canonical cell*. Then function values are sampled at n specially selected points, the *Gauss points*. Each Gauss point is associated with a *weight*. The summation of the products of the function values and the weights are calculated. Finally, the summation is multiplied with the volume of the element to compute the integration:

$$\int_e f(x)\,dx \approx \|V\| \sum_{i=1}^{n} w_i f(x_i),$$



(a) A Tetrahedral Element (b) The Canonical Element

Figure 1: The Canonical Cell of Tetrahedral Element



Gauss points	Coordinates	Weights
a	α, β, β	*0.25*
b	β, α, β	*0.25*
c	β, β, α	*0.25*
d	β, β, β	*0.25*

α=0.58541020 β=0.13819660

Figure 2: Quadratic Order Gauss Points

where $f(x)$ is the function to be integrated, $\|V\|$ is the volume of the element, and x_i and w_i are the coordinates and weights of the Gauss points. The precision of this method depends on the number of the Gauss points used in the integration. A quadratic order accuracy is achieved, if 4 Gauss points are employed. Detail description of Gauss quadrature method can be found in [1, 32]. In Figure 1, a tetrahedral cell and the canonical cell are shown. The locations and weights of the 4 Gauss points of the canonical cell are described in Figure 2.

In our splatting method, the Gauss points of an FEA cell are used as the sampling points in the function reconstruction. The original function, in the cell, is reconstructed by using the following equation:

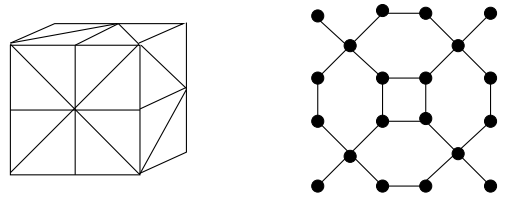
$$f(x) \approx \sum_{i=1}^{n} w_i f(x_i) \phi_i(x),$$

where x_i and w_i are the positions and weights of the Gauss points, and $\phi_i(x)$ are the reconstruction kernels. Since the projection of a $3D$ Gauss function is a $2D$ Gauss function, for convenience, $3D$ Gauss functions are served as the kernels in the function reconstruction.

2.2 Adjacency Graph of Gauss Points

Adjacency graphs are utilized to represent the connectivities of FEA cells [22, 25, 26, 30]. The adjacency graph of an FEA data set is defined by representing each cell by using a graph vertex. If two cells are neighbors, the corresponding graph vertices are connected by an edge. An example of FEA data adjacency graph is depicted in Figure 3. Adjacency graphs can be computed in linear time complexity by using the methods proposed in [22, 24].

Since the adjacency graph is the dual graph of the FEA mesh, it is a good approximation of the mesh topologically. The adjacency graph is modified in this work to obtain better accuracy. Instead of using one vertex, four Gauss points are used to represent a tetrahedral cell. The 4 Gauss points of the cell are connected to form a subgraph with 4 vertices and 6 edges. Then each Gauss point of this subgraph is connected to the nearest Gauss points of the neighboring cells, and the Gauss point adjacency graph is created. A $2D$ example of Gauss point adjacency graph is displayed in Figure 4. The mesh is comprised with 4 triangle cells, and three Gauss points



(a) An FEA Data Set of 20 Cells (b) The Adjacency Graph

Figure 3: Example of FEA Data Adjacency Graph

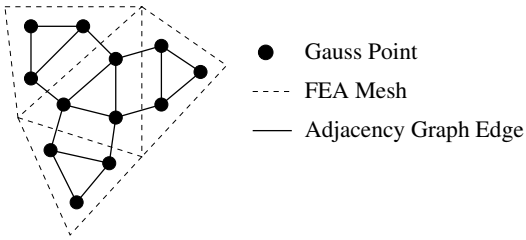


Figure 4: 2D Example of Gauss Point Adjacency Graph

are calculated in each cell. The FEA mesh is depicted with dash lines while the adjacency graph is drawn by using solid lines.

During the creation of the adjacency graph, the attributes of the Gauss points are computed and stored along with the graph. The function value and coordinates of the Gauss points are calculated by interpolating the function values and coordinates of the cell nodes. The gradient is obtained by evaluating the partial derivatives of the interpolation function of of the function value. The influence scope is a sphere centered at the Gauss point. Its radius is equal to 4 times of the distance between the Gauss point and the nearest cell node. This method of computing the influence scope radius is based on our experiments. By using larger influence scopes, the final image would be too opaque and too fuzzy to display the details, while using smaller influence scopes may produce holes in the image. A 2D example of influence scope is illustrated in Figure 5. The Gauss point is displayed by using a rectangle while the cell nodes are represented by using circles.

To compute the diffusion, the difference between the function value of the point and the average function value of the neighboring points is calculated first. Then the difference is divided by the influence scope volume of the point to obtain the diffusion:

$$f_{avg} = \frac{\sum_1^k f_j V_j}{\sum_1^k V_j},$$

$$D_f(x_i) = \frac{\|f_i - f_{avg}\|}{V_i},$$

where f_{avg} is the average function value of the neighboring points, f_j and V_j are the function values and influence scope volumes of the neighboring points, f_i, V_i, and x_i denote the function value, influence scope volume, and coordinates of this Gauss point, and D_f represents the diffusion.

2.3 LoD Representation of FEA Data

Once the adjacency graph is completed, a breadth-first-search is triggered to merge the graph vertices. If the function value difference between a vertex and one of its neighboring vertices is within a predefined threshold, the two vertices are merged to generate a new vertex. The coordinates, function value, gradient, and diffusion of the new vertex are obtained by computing the weighted average of

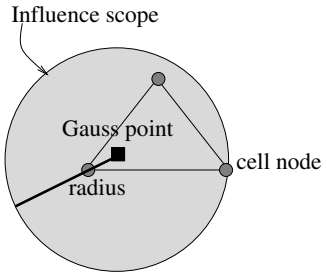


Figure 5: 2D Example of Influence Scope of Gauss Point

these attributes of the two vertices. The influence scope volumes of the two vertices serve as the weights in the computation:

$$g_{new} = \frac{g_1 V_1 + g_2 V_2}{V_1 + V_2},$$

where g_1 and g_2 represent the attributes of the two vertices, g_{new} denotes the attributes of the new vertex, and V_1 and V_2 represent the influence scope volumes of the two vertices. The influence scope volume of the new vertex is calculated by:

$$V_{new} = V_1 + V_2 - V_{1\cap 2},$$

where V_{new} is the influence scope volume of the new vertex, and $V_{1\cap 2}$ is the volume of the intersection of the influence scopes of the two vertices. Since the new influence scope is a sphere, its radius can be determined once its volume is known.

Based on our experiments, the image quality may be too fuzzy if vertices with large influence scopes are created. To avoid this problem, if the the influence scope diameter of the new vertex exceeds a predefined limited, the two vertices will not be merged. The limit of influence scope diameter is defined by:

$$k * \frac{L_d}{\sqrt[3]{N}},$$

where k is a constant decided by users, L_d is the length of the diagonal of the bounding box of the data set, and N is the number of cells. The other skill to improve image quality is to merge vertices whose influence scopes overlapping most. If more than one neighboring vertices satisfying the criteria, the neighboring vertex whose influence scope overlapping with that of the current vertex most is selected in the vertex merging. This skill helps us to avoid losing contrast in the final image.

The breadth-first-search stops when the function value difference between any two neighboring vertices exceeds the threshold. The remaining vertices and the newly created vertices form a new adjacency graph. This new graph is comprised with less vertices and serves as a coarser approximation of the FEA data. Then the threshold is increased, and this new adjacency graph is reduced to construct another adjacency graph. As this graph reduction repeats, multiple adjacency graphs are constructed. The LoD representation is generated. An example of LoD adjacency graph is depicted in Figure 6. The original adjacency graph with 160 vertices is shown in the left part of the figure while the next level of adjacency graph containing 66 vertices is illustrated in the right part.

3 LoD Adjacency Graph Rendering

To visualize the FEA data, users are asked to select a level of adjacency graph. The selected adjacency graph is rendered by using a splatting method. At first, all graph vertices are sorted according to their visibility orders. Then they are projected onto the image plane

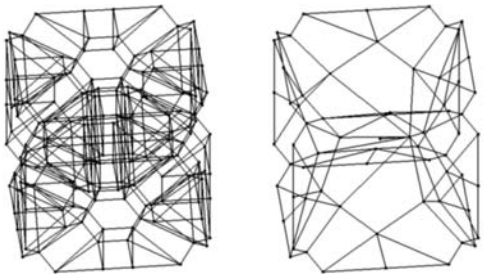


Figure 6: Example of LoD Adjacency Graph

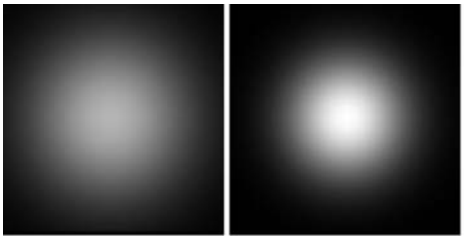


Figure 7: Footprint Function of Different Variances

in back-to-front order. To speed up the projection, the vertices are rendered by using billboards. The billboards are modulated with textures to generate the footprints of the vertices. The footprints are composited to form the final image.

3.1 The Footprint Function and Texture Maps

The splatting methods proposed in [28, 33] are used in our work. Since 3*D* Gauss functions are served as the reconstruction kernels and the parallel projection of a 3D Gauss function is a 2D Gauss function, 2D Gauss functions are used as the footprint functions in blending the effects of the vertices. A 2D Gaussian function is defined by:

$$G(x,y) = S_f * e^{\frac{-(x^2+y^2)}{2\sigma^2}}, \quad (1)$$

where S_f is the scaling factor, and σ is the variance.

Evaluating the footprint function on the grid points of a 2D rectangular table and regarding the footprint function values as opacities, a texture map is constructed. The texture map is used to modulate billboards to produce the footprints of vertices. By using different variances and scaling factors, different texture maps are produced. A small variance results in a texture map with greater variation of opacity. This phenomena is illustrated in Figure 7. The variance of the footprint function in the left image is twice larger than that of the footprint function in the right image. Therefore, the variation of opacity in the right image is more significant. The scaling factor is another factor which influences the texture map. If the scaling factor is increased, the footprint function values are enlarged and the texture map is more saturated. In this work, the initial value of the scaling factor is equal to the weight of the Gauss point. However, its value will be modified according to the size of the influence scope. The method of modifying the scaling factor is presented in the following subsection.

3.2 Diffusion and Influence Scopes

The diffusion of a vertex is an estimate of the variation of the function value. If the diffusion is large, the function value varies significantly near the vertex. To reflect the quick change of function value, the billboard of the vertex is modulated with a texture map of greater variation. On the other hand, if the diffusion is low, the function value varies smoothly. The billboard should be modulated with a texture map of low contrast. A heuristic method is adopted for calculating the variance of the footprint function. At first, a standard variance, σ_{avg}, is selected as the basis. Given a diffusion value, D_f, the variance, σ, is computed by:

$$\sigma^2 = min(\frac{\ln D_{avg}}{\ln D_f}, c)\sigma_{avg}^2,$$

where D_{avg} is the average value of the diffusions of all vertices, and c is a constant defined by users to avoid producing infinite variance when D_f is small.

Beside the diffusion, the volume of influence scope also affects the variance and the scaling factor. The projection of a large influence scope is wider and more opaque. The intensity in the projection also varies more quickly. Therefore the billboard should be modulated with a texture map with more saturated opacity and greater variation. The methods proposed in [12] are modified to adjust the footprint function to generate the desired texture map. First the average diameter of influence scope is calculated and treated as the standard diameter. If the diameter of a influence scope is m times larger than the standard diameter, then the scaling factor and the variance are modified by:

$$S_{new} = 1 - (1 - S_{old})^m,$$

$$\sigma_{new}^2 = \frac{\sigma_{old}^2}{m},$$

where S_{new} and σ_{new} are the new scaling factor and variance, and S_{old} and σ_{old} are the original scaling factor and variance.

3.3 Billboard Orientation and Shapes

In the splatting method, the projection of a vertex is rendered by using a semi-transparent billboard. Theoretically, the billboard is a patch in the iso-surface passing through the vertex, and the gradient is the normal vector of the patch. When the gradient is nearly orthogonal to the view direction, the patch should appear narrow to the viewer. Therefore, the billboard has to be rotated and reshaped. At first, the gradient is projected onto the image plane to generate a 2D vector. Then the billboard is rotated until one of its axes, the y-axis, coincides with the 2D vector. Then the billboard is scaled down in its y-axis. The scaling coefficient,C_f, is computed by:

$$C_f = \max(\frac{\vec{g}}{\|\vec{g}\|} \cdot \frac{\vec{v}}{\|\vec{v}\|}, 0.2),$$

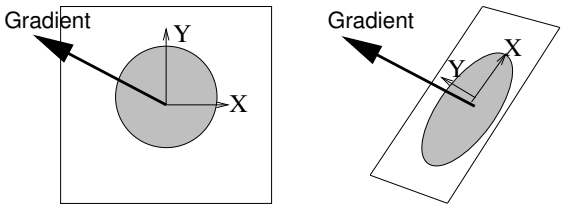
where $\vec{g}$ is the gradient, and $\vec{v}$ is the view direction. The scalar value, 0.2, is treated as a threshold to avoid creating a slim billboard. An example of rotating and reshaping the billboard by using the gradient is demonstrated in Figure 8.

4 Implementation and Test Results

The whole system is implemented on a desk-top machine equipped with a 2.0 GHz cpu and 768M bytes main memory. The operating system is RedHat Linux 7.0. The splatting rendering procedure is implemented by using OpenGL under the Linux system. The image resolution is 700*x*700.

4.1 Implementation Issues

In the construction of the LoD structure, the threshold is linearly increased between every two iterations of the graph reduction. Therefore, in the i-th iteration of graph reduction, the criterion of merging



The Original Billboard

The Billboard after Rotation & Reshaping

Figure 8: Rotating and Reshaping of Billboard

two vertices is determined by:

$$\frac{\|f_1 - f_2\|}{\|f_{max} - f_{min}\|} \leq i * \varepsilon, \tag{2}$$

where f_1 and f_2 are the function values of the vertices, f_{max} and f_{min} are the maximum and the minimum function values of the data set, and ε is a constant. The value, $i * \varepsilon$, is called the *reduction error*, and ε is the *incremental value* of the reduction error. The reduction error represents the relative error introduced in the graph reduction. Given a fixed reduction error, there are two ways of creating an adjacency graph. The first one is to create the adjacency graph by starting with a small reduction error. Then the reduction error is gradually increased and the reduction process is repeated until the reduction error equal to the limit. The second way is to construct the adjacency graph by using the given reduction error directly and performing the graph reduction just once. The reduction errors of the two adjacency graphs are the same. However, based on our experiments, the adjacency graph generated by using the first method results in better image quality than that produced by using the second method.

To speed up the splatting, some approximating techniques are adopted in the implementation. Sixteen texture maps are generated by using different variances at a preprocessing stage. When splatting a vertex, the diffusion and the influence scope diameter are used to compute the variance of the footprint function. The texture map with the nearest variance is selected for the splatting. The influence scope diameter is also used in computing the scaling factor of the footprint function. However, the scaling factor is not directly used to modify the texture map. Instead it is utilized to increase or decrease the opacity of the billboard. Based on the function value of the vertex, the color and opacity of the billboard are retrieved from a look-up table. The billboard size is set to the diameter of the influence scope. The billboard is shaded by using Phong Illumination Model. The gradient is served as the normal vector in the shading computation.

Three data sets are utilized in the experiments. To create the first data set, we split a unit cube into a tetrahedral mesh. The function value of each node is set to the distance from the cube center to the node. This data set is called the *Standard Data Set*. The second test data set contains a wind-tunnel simulation of the M6-wing. The energy values of the data set are rendered in the tests. The third data set is created by releasing particles in a turbulence flow field around a spherical obstacle inside a round tube. The density of particle inside the domain is visualized in the tests.

4.2 Costs of Constructing LoD Representation

For each data set, three adjacency graphs are constructed. The graph reduction is carried out many times to build an adjacency graph. The incremental values of reduction error, ε, for the standard data, the M6 wing data, and the turbulence flow data are 0.000134, 0.00177, and 0.000058 respectively. The reduction errors and the number of iterations of the graph reduction performed to generate the adjacency graphs are displayed in Table 1. The numbers of points in the adjacency graphs are shown in the second column. The reduction errors are displayed in the third column. The last column contains the numbers of iterations of the graph reduction performed. For the standard data, the first adjacency graph and the adjacency graphs produced in the $19th$ and the $36th$ iteration are selected. For the M6 wing data, the original adjacency graph and the adjacency graphs produced in the $11th$ and the $85th$ iteration are kept for the rendering processing. For the turbulence data, the first adjacency graph and the adjacency graphs generated in the $6th$ and the $42nd$ iteration are stored.

Table 1: Errors of Adjacency Graph Reduction

Data Set	*#Point*	*Error*	*#iteration*
Standard	1,703,580	0.0	0
	442,297	0.00255	19
	96,361	0.00482	36
M6 Wing	1,151,848	0.0	0
	493,003	0.01947	11
	199,480	0.09945	85
Turbulence	3,343,008	0.0	0
	982,753	0.00035	6
	596,866	0.00243	42

The costs of constructing the adjacency graphs are depicted in Table 2. The third column contains the reduction ratios. A reduction ratio is computed by dividing the number of points in the first level of adjacency graph by the number of points in the current level of adjacency graph. The last column displays the costs to build each level of the adjacency graph from the first level adjacency graph. The costs are measured in seconds. To build a new adjacency graph, several iterations of graph reduction have to be carried out. For example, the 2nd and the 3rd adjacency graphs of the M6 wing data are created by performing 11 and 85 iterations of graph reduction. Dividing the cost by the number of iteration, the average cost of performing one iteration of the graph reduction is computed. The average graph reduction costs of the standard data, the M6 wing data, and the turbulence flow data are 3.98, 1.16, and 10.24 seconds respectively. These costs are influenced by the sizes of the data sets. The average graph reduction cost of a larger data set is always bigger. The topology of mesh is also a key factor. The topological structures of the standard data and the M6 wing data are simpler. Therefore, their average graph reduction costs are lower. The main memory capacity is another important issue. In our implementation, the whole LoD structure is kept in the main memory when performing the graph reduction. The required memory space of the turbulence flow data exceeds the memory capacity of the computer. Page-swapping is triggered by the OS to transfer data between the disk and the main memory. Therefore the average reduction cost is increased.

4.3 Costs of Splatting Rendering

The costs of rendering the LoD structures of the test data are collected and displayed in Table 3. The costs, measured in seconds, are displayed in the third column. The speed-up rates, contained in the last column, are obtained by dividing the rendering cost of each level of the adjacency graph with the rendering cost of the first level adjacency graph. The standard test data reveals an obvious speed-up in the rendering computation as the number of points decreased. However, the speed-up for the other two test data sets is lower than our expectation. The reason is that the view point is put inside the domains, and many vertices are outside the view volume and culled

Table 2: Costs of Constructing LoD Representation

Data Set	*#Point*	*Reduction Ratio*	*Costs(sec.)*
Standard	1,703,580	1:1	0.0
	442,297	3.85:1	61.76
	96,361	17.68:1	111.05
M6 Wing	1,151,848	1:1	0.0
	493,003	2.34:1	25.49
	199,480	5.77:1	98.22
Turbulence	3,343,008	1:1	0.0
	982,753	3.40:1	254.05
	596,866	5.60:1	430.12

Table 3: Costs of Volume Rendering

Data Set	*Num. of Point*	*Costs(sec.)*	*Speed-up*
Standard	1,703,580	30.87	1.0
	442,297	8.08	3.82
	96,361	2.44	12.65
M6 Wing	1,151,848	35.70	1.0
	493,003	22.23	1.61
	199,480	16.16	2.21
Turbulence	3,343,008	37.17	1.0
	982,753	25.50	1.46
	596,866	15.84	2.38

by the graphics pipeline. Therefore, only a portion of the vertices are involved in the rendering, and the speed-up is not linear.

4.4 Visualization Results

The images shown in Figures 9,10, and 11 are the visualization results of the first test data. The original adjacency graph contains more than 1.7 million of points. The volume rendering image of this graph is shown in Figure 9. The other two adjacency graphs contain about 442 and 96 thousand points respectively. The volume rendering images of these graphs are shown in Figures 10 and 11. Comparing these two images with Figure 9, the quality of image is still clear enough to reveal the distribution of function value, even though the data size is reduced by more than 17 times. The next series of pictures displays the visualization results of the M6-wing data. The distribution of energy around the wind is displayed. The original adjacency graph is composed of about 1.15 million of points. The other two coarser adjacency graphs are comprised with 493 and 199 thousand points. The volume rendering images of these adjacency graphs are shown in Figures 12, 13, and 14. As the data size is reduced by more than 82%, the image of the adjacency graph with lowest resolution still sustains an accurate approximation of the volume rendering image of the original adjacency graph. The third series of pictures shows the volume rendering images of the turbulence flow data. The colors are used to show the density of particles traveling in the flow field. In Figure 15, the original adjacency graph of 3.3 million points is rendered. The second level of adjacency graph consists of 997 thousand points. The volume rendering image of the graph is displayed in Figure 16. The adjacency graph of the third level contains only 599 thousand vertices. The image of Figure 17 is the visualization result of the adjacency graph. Comparing the three pictures, it is obvious that the image produced by rendering the simplified adjacency graph still reveals the key features of the flow field, though the number of vertices is reduced to about one sixth of the original data.

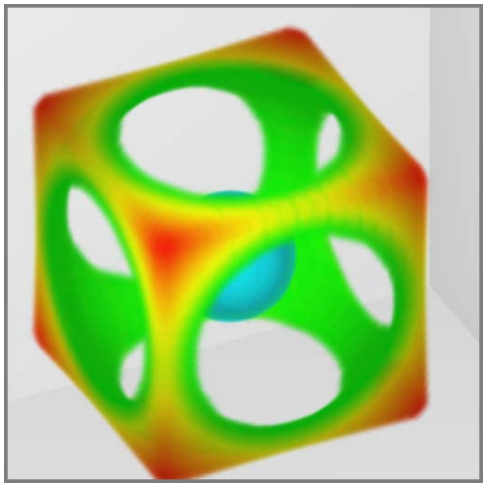


Figure 9: Standard Data Set of 1.7 M Points

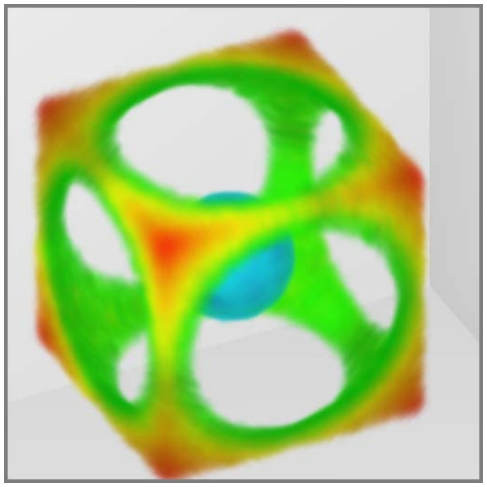


Figure 10: Standard Data Set of 442K Points

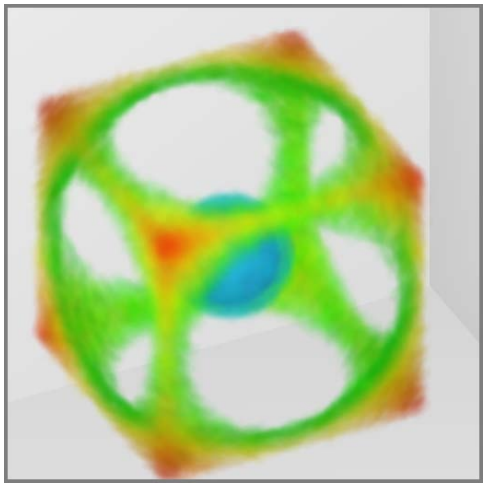


Figure 11: Standard Data Set of 96K Points

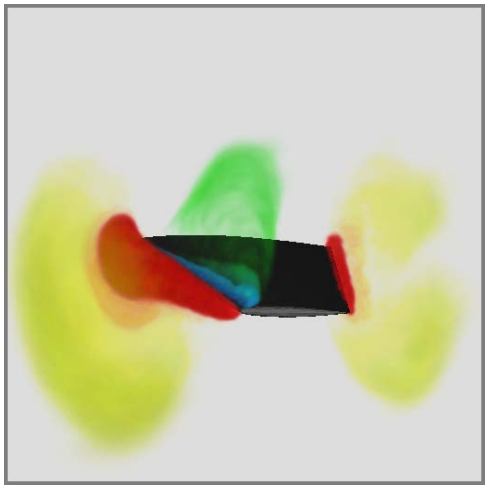


Figure 12: M6 Wing Data Set of 1.15 M Points

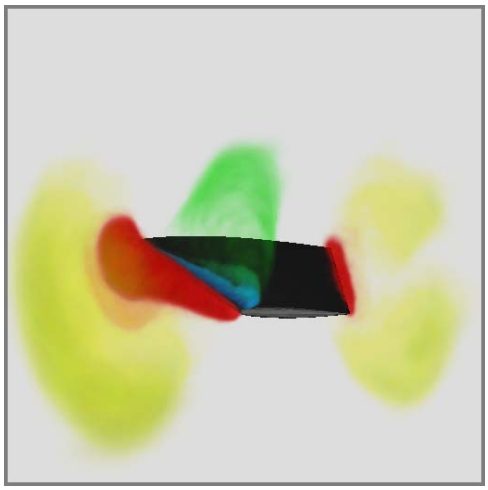


Figure 13: M6 Wing Data Set of 493K Points

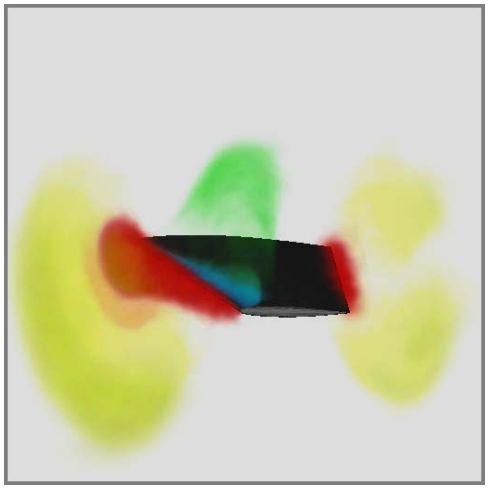


Figure 14: M6 Wing Data Set of 199K Points

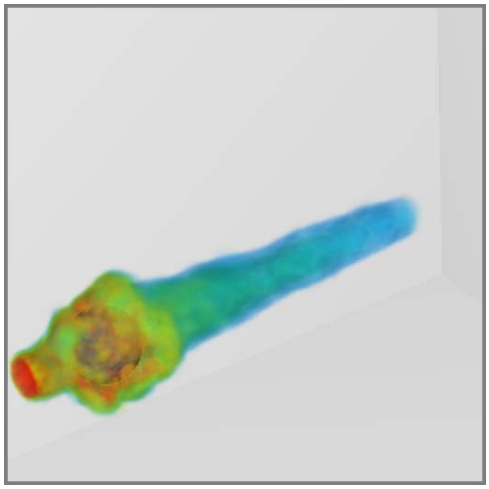


Figure 15: Turbulence Flow Data Set of 3.3 M Points

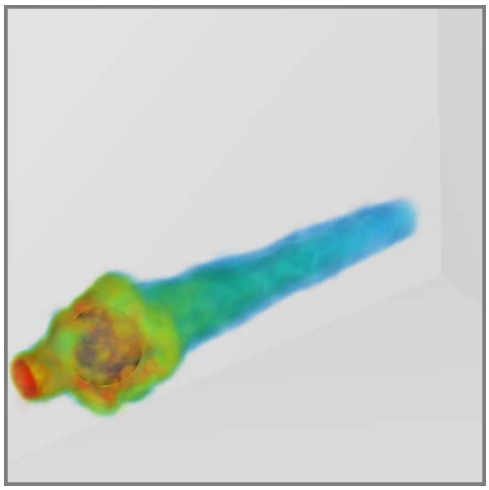


Figure 16: Turbulence Flow Data Set of 982K Points

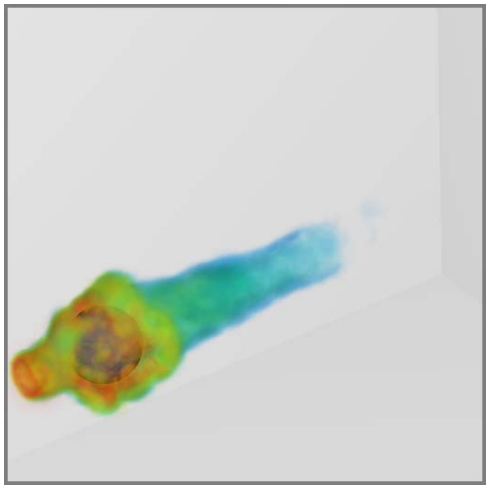


Figure 17: Turbulence Flow Data Set of 596K Points

5 Conclusion and Future Work

In this article, an LoD volume visualization method for FEA data is presented. It constructs hierarchical representations of FEA data, and allow users to render the data with different precisions. Furthermore, our method is a meshless method. It can be extended to cope with scattered scientific data and the data generated by using meshless FEA computational methods. Some FEA computation may last for a long time before the final solution is generated. It is useful to visualize the partial results while the computation is carrying on such that ill-modeled computing can be terminated before wasting too much resources. Our method can be modified to combined with FEA solvers to generate volume rendering images during the computing. Therefore, the progression of the computing can be monitored interactively. Data compression is another important issue for FEA computing [23, 31]. The adjacency graph structure offers an alternative choice for compressing FEA data. Our LoD representation can be utilized to replace the mesh of FEA data to save memory space.

References

[1] D. Burnett. Finite Element Analysis. 1987.

[2] Prashant Chopra and Joerg Meyer. TetFusion: An Algorithm for Rapid Tetrahedral Mesh Simplification. In *Proceedings of the IEEE Visualization'2002*, pages 133–140, 2002.

[3] p. Cignoni, D. Costanza, C. Mantani, C. Rocchini, and R. Scopigno. Simplification of Tetrahedral Meshes with Accurate Error Evaluation. In *Proceedings of IEEE Visualization'2000 Conference*, August 2000.

[4] Paolo Cignoni, Claudio Montani, Enrico Puppa, and Roberts Scopigno. Multiresolution Representation and Visualization of Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 11:352–369, 1997.

[5] A. Roger Crawfis and Nelson Max. Texture Splats for 3D Scalar and Vector Field Visualization. In *Proceedings of the IEEE Visualization'93*, 1993.

[6] R. Gallagher and J. Nagtegaal. An Efficient 3D Visualization Techniques for Finite Element and Other Coarse Volume. *ACM Computer Graphics*, 23:185–193, 1989.

[7] M. P. Garrity. Raytracing Irregular Volume Data. *Computer Graphics*, 24:35–40, 1990.

[8] C. Giertsen and J. Petersen. Parallel Volume Rendering on a Network of Workstations. *IEEE Computer Graphics and Applications*, 13:16–23, 1993.

[9] Wei Hong and Arie Kaufman. Feature Preserved Volume Simplification. In *Proceedings of the 8th ACM Symposium on Solid Modeling and Applications*, pages 334–339, 2003.

[10] James Kajiya. Ray Tracing Volume Densities. In *Proceedings of SIGGRAPH'84 Conference*, pages 165–174, 1984.

[11] Eric LaMar, Hamann Bernd, and Kenneth Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *Proceedings of IEEE Visualization'1999 Conference*, pages 355–362, 1999.

[12] David Laur and Pat Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. In *Proceedings of SIGGRAPH'91 Conference*, July 1991.

[13] K. L. Ma, J. Painter, and M Krogh. A Data Distributed Parallel Algorithm for Ray-tracing Volume Rendering. In *Proceedings of Parallel Rendering Symposium'93*, pages 15–22, 1993.

[14] Nelson Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[15] Nelson Max, P. Hanrahan, and R. Crawfis. Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. *ACM Computer Graphics*, 24:27–33, 1990.

[16] Jeremy Meredith and Kwan-Liu Ma. Multiresolution View-Dependent Splat Based Volume Rendering of Large Irregular Data. In *Proceeding of IEEE Visualization'2001 Conference*, pages 93–100, 2001.

[17] Klaus Mueller and Roger Crawfis. Eliminating Popping Artifacts in Sheet Buffer Based Splatting. In *Proceedings of the IEEE Visualization '98*, pages 239–245, 1998.

[18] Klaus Mueller and Roger Crawfis. Splatting without the Blur. In *Proceedings of the IEEE Visualization '99*, August 1999.

[19] Peter Shirley and Allan Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. In *Proceedings of 1990 Workshop on Volume Visualization*, pages 63–70, December 1990.

[20] Issac J. Trotts, Bernd Hamann, and Kenneth I. Joy. Simplification of Tetrahedral Meshes with Error Bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):99–108, 1999.

[21] Shyh-Kuang Ueng and Kris Sikorski. Parallel Visualization of 3D Finite Element Analysis Data. In *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing*, pages 808–813, 1995.

[22] Shyh-Kuang Ueng and Kris Sikorski. A Note on a Linear Time Algorithm for Constructing Adjacency Graphs of 3D FEA Data. *The Visual Computer*, 12(9):445–450, 1996.

[23] Shyh-Kuang Ueng and Kris Sikorski. A Data Compression Method for Tetrahedral Meshes. In *Proceedings of SPIE Visualization and Data Analysis 2002*, volume 4665, pages 134–141, San Jose, CA, USA, 2002.

[24] Shyh-Kuang Ueng and Kris Sikorski. An Out-Of-Core Method for Computing Connectivities of Large Unstructured Meshes. In *Proceedings of the 4th Eurographics Workshop on Parallel Graphics and Visualization*, pages 97–103, Blaubeuren, Germany, September 2002.

[25] Shyh-Kuang Ueng, Kris Sikorski, and Kwan-Liu Ma. Fast Algorithms for Visualizing Fluid Motion in Steady Flow on Unstructured Grid. In *Proceedings of IEEE Visualization'1996 Conference*, pages 313–319, 1996.

[26] Shyh-Kuang Ueng, Kris Sikorski, and Kwan-Liu Ma. Out-Of-Core Streamline Visualization on Large Unstructured Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):100–108, 1997.

[27] Lee Westover. Interactive Volume Rendering. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, May 1989.

[28] Lee Westover. Footprint Evaluation for Volume Rendering. In *Proceedings of SIGGRAPH'90 Conference*, August 1990.

[29] Peter L. Williams. Interactive Splatting of Nonrectilinear Volumes. In *Proceedings of IEEE Visualization'1992 Conference*, pages 37–44, 1992.

[30] Peter L. Williams. Visibility Ordering Meshed Polyhedra. *ACM Trans. on Graphics*, 11(2):103–126, 1992.

[31] Chuan-kai Young, Tulika Mitra, and Tzi-cker Chiueh. On-the-Fly Rendering Of Losslessly Compressed Irregular Volume Data. In *Proceedings of the IEEE Visualization'00*, pages 101–108, 2000.

[32] O. C. Zienkiewicz and R. L. Taylor. The Finite Element Method. 1989.

[33] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA Volume Splatting. In *Proceedings of IEEE Visualization'2001*, pages 29–36, 2001.