# Progress on STEP Visualization
# NETL / WVU NARA project

Tim McGraw, Takamitsu Kawai

December 2008

## 1 Personnel

- Tim McGraw, Asst. Professor
  Duties : Research and develop algorithms for information visualization, manage research assistants.
- Takamitsu Kawai, Graduate Research Assistant
  Duties : Research, develop and implement algorithms for information visualization.

## 2 Travel in 2008

- October 19 - October 24 2008; Columbus, Ohio
  Purpose : Attend IEEE Visualization conference to learn state-of-the-art in information visualization and interface with experts in the field.

## 3 Overview

The field of information visualization is still in its infancy compared to scientific visualization, but it is growing rapidly. Practitioners in this field seek to develop visual representations for non-numerical data which give insight and convey meaning in user-friendly and intuitive ways. The difficulty of this problem grows when the datasets are large, high-dimensional or time-varying. The approach we propose for visualization of STEP data has 2 defining characteristics:

1. **GPU implementation.**
   We propose to use the graphics processing unit (GPU) to not only display images, but to compute the intermediate results necessary for visualization. This permits us to exploit the parallel processing power of the GPU and maintain interactivity of the application. This is difficult to achieve in other parallel processing environments. The growing memory on recent GPUs allows us to apply our algorithms to large scale datasets.

2. **Casting discrete entities into the continuous domain.**
   Many existing information visualization techniques utilize structures which are discrete in nature - tables, bar charts, line graphs. However, interacting with this data may introduce concepts which are continuous in nature. One example of such a concept is scale. A large dataset is commonly simplified by clustering. Entities which are similar according to some criteria are collapsed into a single entity. Clustering can be applied recursively so that similar clusters can also be joined together. At a very coarse scale a dataset can be represented as a few large clusters, but at a finer scale the individual entities may emerge. When presenting a user two views of the data at 2 different scales it can be difficult to identify which entities at the finer scale are related to the clusters at the coarser scale. By allowing the user to select the scale in a continuous fashion we may eliminate much ambiguity for visualization of large datasets. Continuous visualization permits improved retention of the relationships between entities and their clusters.

These characteristics will lead to interactive methods for large graph visualization and annotation. Continuous representations are are also more natural to store and manipulate on the GPU since it is designed to manipulate and display triangle meshes and raster images.

## 4 Introduction

As the complexity of graphs increases, it becomes more important to develop sophisticated graph visualization techniques that produce comprehensible results. Here we discuss our novel continuous level of detail (CLOD) techniques for visualizing inclusive relations among clusters in graphs.

Researchers have developed various graph layout/visualization libraries such as: OGDF (Open Graph Drawing Framework) [8], Graphviz [11][1], and VTK (The Visualization Toolkit) [3]. We intend to utilize these libraries in our application as the geometric problem of graph layout is beyond the scope of this project. One of the missing features in these graph visualization libraries is the ability to comprehensibly visualize the inclusive relations among clusters in a graph. Generally graphs represent network structures of multiple components. One of the important aspects of such graphs is that users can define multiple clusters within them representing relations between vertices. A cluster typically represents a certain meaningful grouping of graph vertices such as subassemblies in an assembly. The clustering may be done interactively by users or automatically by some clustering algorithm. These clusters can also be nested - it can contain vertices and/or smaller clusters as its substructure. The important fact is that these inclusive relations among clusters are orthogonal to the original network structure of the graph. We would like to clearly visualize the transition of the formation of these clusters. The conventional graph visualization libraries typically use dots or small icons to represent vertices and use lines or curves for edges. They also allow users to specify clusters of the vertices and produce clustered layouts taking into account the clustering information. The clusters are generally represented as nested boxes in the output. Although technically these nested boxes themselves represent the inclusive relations among clusters, it is difficult to see the process of the formation of those nested structures.

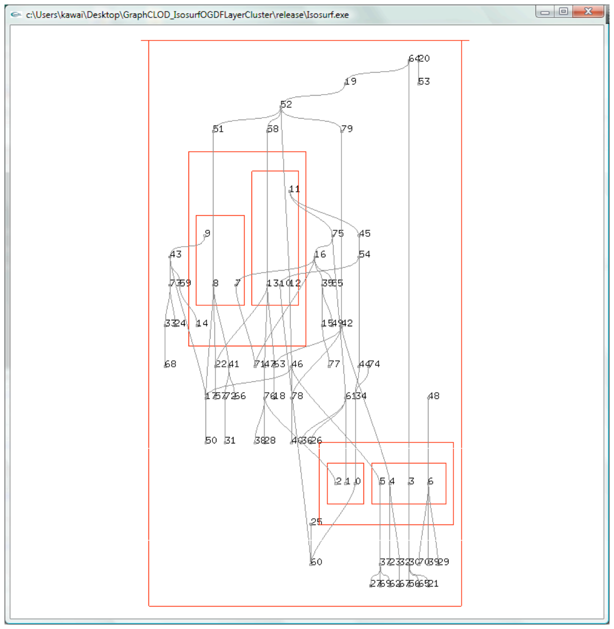
c:\Users\kawai\Desktop\GraphCLOD_IsosurfOGDFLayerCluster\release\Isosurf.exe

Figure 1: A clustered graph. Clusters are represented as nested boxes.

Figure 1 shows an example of a clustered graph and Figure 2 shows an alternative to the nested box scheme. Here the nodes collapse together and merge smoothly as they overlap. This is achieved by representing the nodes not as individual discrete circles, but as the isocontours of a continuous function. The cluster is rendered as a single node whose attributes are a combination of the attribute of the constituent nodes. In this case we have used color to represent node attributes, but texture and other visual attributes will be incorporated in the future. The appearance of the cluster is computed on the GPU and the animation from the first to last frame of Figure 2 is smooth and fast in the running application. By rendering the cluster in this compact way we can efficiently draw larger and more complex graphs.

Generally, the underlying graph can have any topological structure. In the examples presented here the graph is a directed acyclic graph (DAG). The clustering information can be given either by users or by some clustering algorithms. The layout in Figure 1 is done by the OGDF [8] graph layout library. The black dots with vertex numbers and curves indicate the original underlying DAG. The red boxes indicate clusters. Note that the inclusive relations among clusters is orthogonal to the original DAG structure. The layout library performs the lay-

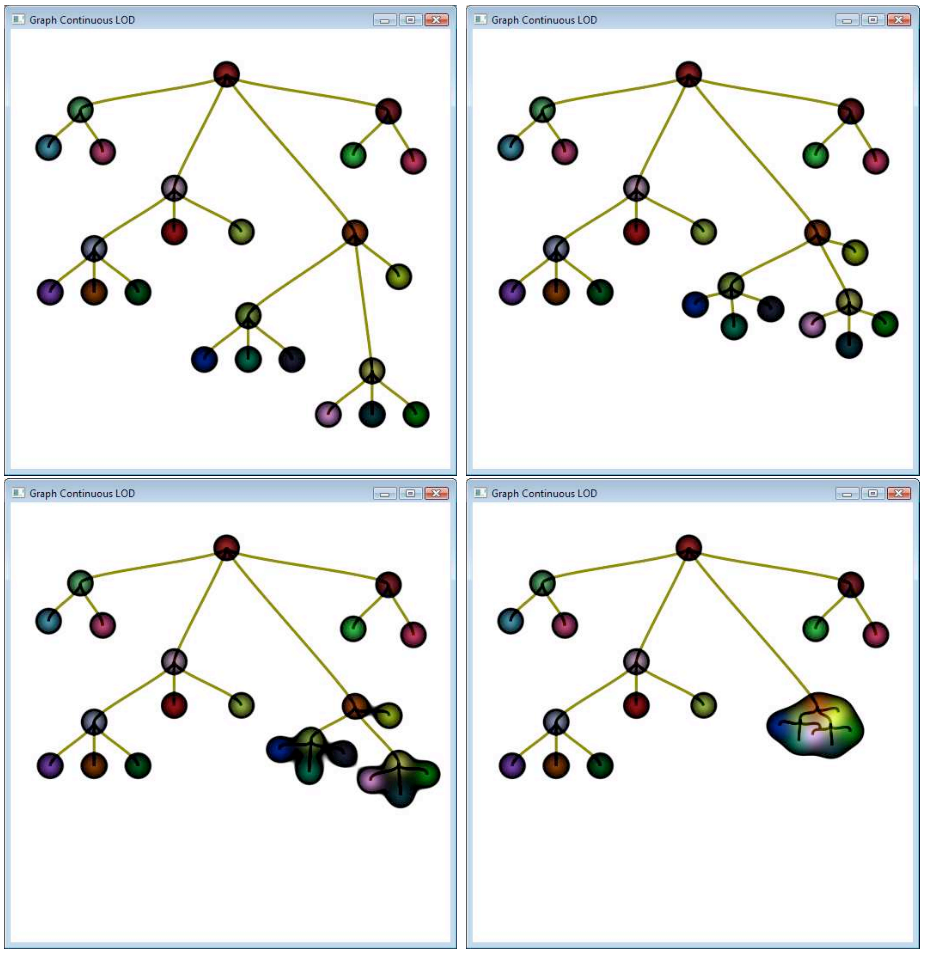

Figure 2: Animation frames demonstrating the transition between individual nodes and a cluster in a continuous manner. The merging of nodes is computed on the GPU.

out taking into account the given clustering information so that the vertices within a cluster are included in the same box yet the original underlying graph structure is kept. In this figure, although we can recognize that there are multiple nested clusters, it is difficult to capture how each one of the clusters fuses together and the fact that the smaller clusters gradually form larger clusters. In order to address this problem, we introduce the concept of continuous level of detail for graphs. Our basic idea is to visualize this cluster formation by utilizing sets of isosurfaces that span multiple clusters in different levels of detail.

## 5 Graph CLOD Visualization Techniques

### 5.1 Input Files

To demonstrate our method, we use STEP files as input data. Figure 3 shows an example visualization of the graph derived from a STEP file. It represents the lower rudder arm of a large ship. The leaf nodes correspond to the most primitive entities such as position and directional vectors. As we go up the hierarchy, higher-level entities such as edges, faces and solids are represented. Since one lower-level entity can be referenced by multiple higher-level entities, generally the entire graph forms a DAG. We used a reduced version of this file in its number of vertices to examine our methods.

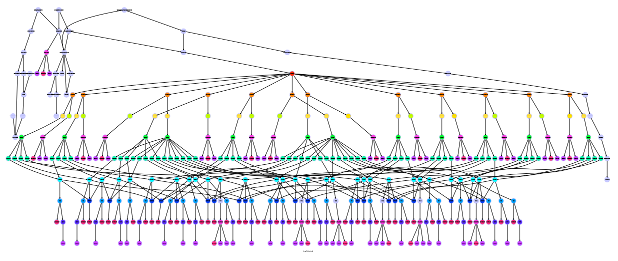

Figure 3: Example of visualization of a STEP file using conventional graph layout library.

### 5.2 Constructing Clustered Graphs

To facilitate our experiments, we developed a parser that extracts the graph structure from an input STEP file. The extracted graphs are assumed to be a DAG and stored as a graph object within the graph layout library. We used the clustered graph layout algorithm available in OGDF. It allows users to specify clustering information and perform layouts using this information. Figure 1 shows an example of clustered layout.

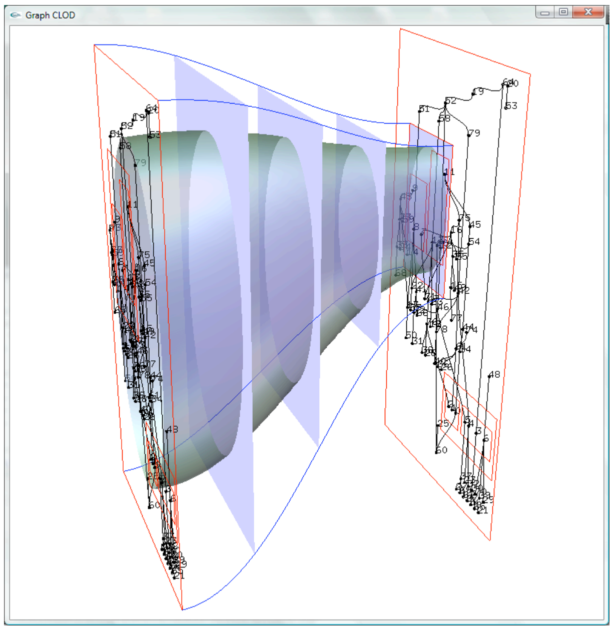

Figure 4: Continuous transition between clusters.

## 5.3 Constructing Isosurfaces

The number of levels of detail $N$ is predetermined by users and we create one layer per one level of detail. A layer is a container of the clustered layout of the original input graph. Once the clustered graph layout is performed, the layout information is copied into all the layers. In each layer, the bounding box of each cluster is determined by the layout algorithm. We create isosurfaces based on the coordinates of these bounding boxes. One set of isosurfaces is created between two adjacent layers $L_n$ and $L_{n+1}$. Therefore for $N$ levels of detail there are $N$ corresponding layers and $N-1$ sets of isosurfaces. Figure 5 shows an example of multiple layers. In this figure $N=3$, so 2 isosurfaces will be created. The sets of isosurfaces allow us to visualize how the clusters converge or diverge as we navigate through the different levels of detail. In order to construct isosurfaces, each scalar field is initialized as follows: Let the set of bounding boxes in the $n$-th layer $L_n$ be $B_n$. Denote the $i$-th cluster in $L_n$ as $c_{n,i}$ and its corresponding bounding box as $b_{n,i}$. According to the inclusive relations among clusters, appropriate clusters in adjacent layers are connected. Consider connecting a cluster $c_{n,i}$ and another cluster $c_{n+1,j}$. A smooth transition is made for the corresponding bounding boxes $b_{n,i}$ and $b_{n+1,j}$ by Hermite interpolation [5]. Figure 4 shows an example of the transition between the bounding boxes $b_{n,i}$ and $b_{n+1,j}$. Each one of four corners of the bounding boxes are interpolated. We set up local coordinate systems for each pair of $L_n$ and $L_{n+1}$. $x$ and $y$

coordinates are set up so that the plane of layers is parallel to the $xy$ plane. Thus $z$ direction is perpendicular to the layer planes. The Hermite interpolation is given by

$$\begin{aligned}
h_1 &= 2s^3 - 3s^2 + 1 \\
h_2 &= -2s^3 + 3s^2 \\
h_3 &= s^3 - 2s^2 + s \\
h_4 &= s^3 - s^2 \\
\boldsymbol{p} &= h_1\boldsymbol{p}_0 + h_2\boldsymbol{p}_1 + h_3\boldsymbol{t}_0 + h_4\boldsymbol{t}_1
\end{aligned}$$

where $0 \le s \le 1$ is a parameter, $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ are the three-dimensional coordinates of one of four corner points of $b_{n,i}$ and $b_{n+1,j}$, respectively. $\boldsymbol{t}_0 = \boldsymbol{t}_1 = (0, 0, 1)$ are the tangent vectors of the curve at the start and end points of the curve. Once the trajectory of the curve is determined, radial basis functions (RBFs) [16][10][6][15][14] based on a Gaussian kernel are accumulated into the scalar field along the curve. For a point $\boldsymbol{p} = (p_x, p_y, p_z)$ on the curve the RBF is given by

$$s(\boldsymbol{x}(x, y, z); \boldsymbol{p}) = w(p_z) \exp\left(-\alpha\left(\left(\frac{x - p_x}{a}\right)^2 + \left(\frac{y - p_y}{b}\right)^2\right)\right)$$

where $w$ is a weight function, $\alpha$ is a coefficient that determines the steepness of the kernel, and $a$ and $b$ is the half the size of the interpolated bounding box. The $w(z)$ controls the strength of the kernel depending on the $z$ position. Since multiple clusters from $L_n$ can be fused into to one cluster in $L_{n+1}$, $w$ needs to be determined so that the strength of each cluster connection is equally distributed based on the number of incoming/outgoing connections for each cluster. Let the multiplicity at the cluster $c_{n,i}$ be $M_0$ and at the cluster $c_{n+1,j}$ be $M_1$. The weight is given by

$$w(z) = (1 - z)\frac{1}{M_0} + z\frac{1}{M_1}$$

where $z$ is the parameter ranging from 0 (at $c_{n,i}$) to 1 (at $c_{n+1,j}$) that parameterizes the $z$-position of a point on the Hermite curve.

### 5.4 Rendering Isosurfaces

After scalar fields are constructed for each pair of layers, isosurfaces are extracted from each scalar field. Our renderer uses the marching tetrahedra algorithm [13][4] which is a variant of the marching cubes algorithm [7][4]. Our implementation of the algorithm is GPU-based and it produces triangles using the geometry shader. For lighting we used the per-pixel Phong lighting to give the appearance of a smooth surface. In future work, spatially varying material properties will be used for indicating data attributes. Figure 6 shows the rendering result. The entire set of isosurfaces represent the continuous level of detail making a smooth transition over multiple layers connecting clusters in each layers. We can clearly identify the inclusive relations among clusters by tracing the branching of the surfaces.

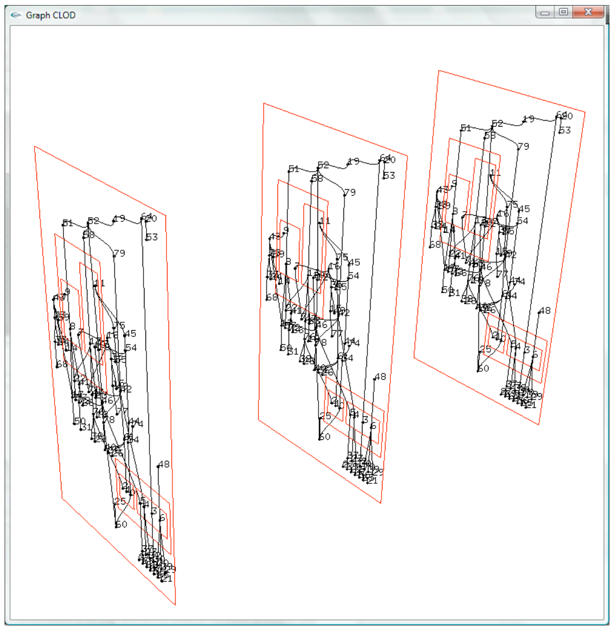

Figure 5: Clustered graph (multiple layers).

## 5.5 Issues in Parsing STEP Files

We attempted to use the STEP importer available in Open CASCADE library [2], however we found some difficulties in applying the library when we tried to extract the underlying graph structure. Although it allows us to traverse the topological data structure such as vertices, edges, faces, and faces, it does not completely reveal the original graph structure included in the STEP file. Therefore we decided to develop our own STEP file parser. Although this parser is a simple one, it has enough capability to extract all the vertices/edges relations included in the input STEP file.

## 6 Conclusions and Future Work

We have described our novel continuous level of detail visualization techniques for clustered graphs. The methods we have developed are amenable to implementation on the GPU. We have achieved interactive framerates for all examples shown in this report.

Our future work will include augmenting the surface shown in Figure 6 with features such as text annotation, texture mapping and bump mapping in order to provide the user with more information about the nodes and clusters being represented.

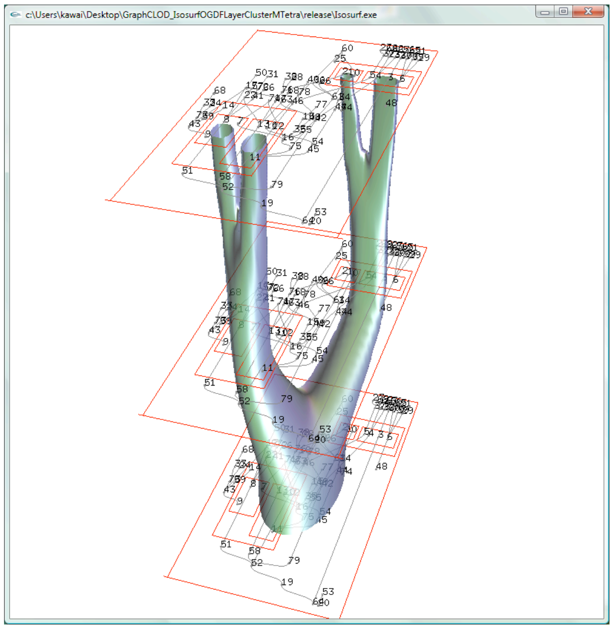
c:\Users\kawai\Desktop\GraphCLOD_IsosurfOGDFLayerClusterMTetra\release\Isosurf.exe

Figure 6: Rendering across multiple scales. The hierarchical structures of the clusters is clearly visible. Boxes are shown overlaid on the surface for reference only.

We are also working on other graphics techniques to allow users to manipulate vertices and clusters more interactively. An approach using vector fields to impose repulsive forces between nodes which are not in the same cluster (to prevent them from merging) can permit the user to dynamically drag nodes and rearrange the graph. A prototype system which computes these fields on the GPU currently in development.

Currently we depend on a conventional isosurface extraction method to produce the surface in Figure 6. We are developing more sophisticated methods for constructing this isosurface by efficiently computing the topological structure using Reeb graphs [9][12]. Knowledge of the topology of the surface prior to surface extraction will permit a more rapid surface extraction and allow the surface to be more efficiently texture mapped and represented in the GPU memory.

## References

[1] Graphviz Graph Visualization Software. `http://www.graphviz.org`.

[2] Open CASCADE. `http://www.opencascade.org`.

[3] VTK The Visualization Toolkit. `http://www.vtk.org`.

[4] P. Bourke. Polygonising a Scalar Field. `http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/`.

[5] R. L. Burden and J. D. Faires. *Numerical Analysis*. Brooks/Cole.

[6] J. C. Carr, W. R. Fright, and R. K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging*, 16:96–107, 1997.

[7] S. L. Chan and E. O. Purisima. A new tetrahedral tesselation scheme for isosurface generation. *Computers & Graphics*, 22(1):83 – 90, 1998.

[8] M. Chimani, C. Gutwenger, M. Jünger, K. Klein, P. Mutzel, and M. Schulz. The Open Graph Drawing Framework. In *15th International Symposium on Graph Drawing 2007, Sydney (GD07)*, 2007.

[9] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in Reeb Graphs of 2-Manifolds. *Discrete Comput. Geom.*, 32:231–244, 2004.

[10] T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 67–76. Springer, 2001.

[11] E. R. Gansner and S. C. North. An Open Graph Visualization System and Its Applications. *Software - Practice and Experience*, 30:1203–1233, 1999.

[12] J. Snoeyink H. Carr and U. Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24:75–94, 2002.

[13] W. E. Lorensen and H. E. Cline. Marching Cubes: a high resolution 3D surface reconstruction algorithm. In *Computer Graphics*, volume 21, pages 163–169 (Proc. of SIGGRAPH), 1987.

[14] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel. Multi-level Partition of Unity Implicits. In *SIGGRAPH*, pages 463–470, 2003.

[15] W. Qiang, Z. Pan, C. Chun, and B. Jianjun. Surface Rendering for Parallel Slices of Contours from Medical Imaging. 9:32–37, Jan/Feb., 2007.

[16] G. Turk and J. F. O'Brien. Shape transformation using variational implicit functions. In *SIGGRAPH '99*, pages 335–342, 1999.