DGCVis: An Exploratory 3D Visualization of Graph Pyramids

Andreas Kerren, Florian Breier, and Philip Kügler
Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
kerren@acm.org

Abstract

Hierarchies of plane graphs, called graph pyramids, are used for collecting, storing and analyzing geographical information based on satellite images or other input data. Important for the better understanding of the geographical data, like landscape properties or thematical maps, is the appropriate visualization of graph pyramids and related information. In this paper, we present an interactive visualization tool that supports several coordinated views on graph pyramids, subpyramids, thematical maps, etc. Furthermore, some implementation details and application results are discussed.

1. Introduction

One important research area in Geography, especially Landscape Ecology, is the automatical determination of landscapes and their properties. Such properties can be very complex and interdependent, for example warmth, humidity, or growth of plants in a special area. Moreover, a landscape itself consists of several objects, like lakes or forests. To determine the type of a landscape, we also need information about the size, the shape, as well as some context data of these landscape objects. So, a small forest with streets within a congested urban area could be a recreation area close to a town, whereas a bigger forest without any streets in alpine scenery could be an interesting wildlife habitat.

In the project GEOGRAPH[1], the automatical determination of primitive landscape elements and types is done by segmentation on the basis of images (e.g. satellite images). But conventional segmentation and classification methods, well-known in the Pattern Recognition community (e.g. [3, 11]), do not suffice to solve the problem of composed landscape objects described above because of its hierarchical solution space. What we need is an additional second-order segmentation, i.e., a meta-segmentation, to analyze the input image. It is easy to see that we obtain a whole hierarchy of segmentations with several levels if we continue with segmentation from level to level. Here, graph-based segmentation techniques use node and edge information to perform this meta-segmentation: Each object (landscape area) in a level can be regarded as node in a graph. Edges between nodes indicate the neighborhood relations within that level. The result is a graph hierarchy, called *graph pyramid* [10], which we can use as data structure for retrieving geographical information. Each level in a graph pyramid represents a kind of abstract map (land cover maps) of the input image: Lower pyramid levels describe more concrete maps and higher pyramid levels represent a higher level of abstraction. In the following, we call these maps *thematical maps* due to the possibility to generate them using additional geographical information.

The visualization of graph pyramids and the derived geographical information is part of the GEOGRAPH project and topic of this paper that is organized as follows: In the next Section 2, we shortly discuss some background information about graph pyramids and motivate the importance of visualization. Our visualization approach is presented in Section 3, 4 and 5 including the description of used views and implementation details. In Section 6, some application examples of our tool are shown. Section 7 gives a conclusion and outline of future work.

2. Graph Pyramids

Graph pyramids store hierarchies of graphs and the linkage between consecutive graph levels. They can be generated from images or pre-segmented graphs in GML format [9] extended with additional information. In case of images, each node of the base level represents a pixel with a specific RGB value. This is the node's attribute. The edges form a regular grid. In case of pre-segmented input graphs,

1 This research has been supported by the Austrian Science Fund (FWF) under grant number P14462.

Proceedings of the 2nd International Conference on Coordinated and Multiple Views in Exploratory Visualization, CMV 2004, London, UK, 2004.

the base level is built by the elements of the input graph. Nodes and edges can be associated with attributes, for example color, a path per node for the specification of the area shape that is represented by the node, or in case of edges the degree of border precision.

2.1. Dual Graph Contraction

The meta-segmentation process is performed by the Dual Graph Contraction Library (DGClib). By contracting edges within one level of the pyramid an algorithm, called *Dual Graph Contraction* (DGC), computes the graph of the next higher level. Edge contraction is a process of collapsing the edge $e = (u, v)$ and melting the nodes u and v, i.e. one node (the surviver) remains and the other node is removed. The surviver adopts the edges that were incident to the removed node and its attributes are a combination of those from u, v, and e. This construction method yields geographical information on different abstraction levels. DGC is specified by disjoint *contraction kernels* (trees), each defining a possibly empty set of non-surviving nodes and the surviver. The selection of a contraction kernel depends on edge and node attributes. They are described by so-called *contraction rules* which are highly dependent on the application area. The identification of meaningful contraction rules is part of the GEOGRAPH project and ongoing research.

During a DGC run the nodes are combined to new nodes in higher levels using the contraction rules, i.e., single pixels are identified as parts of bigger objects. If the input image shows a mountain the algorithm finds all single objects on it and places them as nodes in the pyramid (in best case).

More precisely, each level of the pyramid consists of a pair of dual graphs. One graph is called point graph (the graph discussed above) and the other is called face graph; it describes the borders of the areas. Thus, the DGC algorithm is divided into two phases: dual edge contraction and dual face contraction. See the article of Kropatsch [10] for further details on duality and DGC. In the rest of this paper, we consider only point graphs.

The schematic diagram in Figure 1 shows a small graph pyramid with three levels. Directed edges represent the contraction processes, white nodes are collapsed in the next upper level, and dotted edges between the levels symbolize together with the connected nodes all contraction kernels. The visualization of these contraction kernels is an important issue because they illustrate the core of the segmentation process.

2.2. Why Visualization?

The answer to this question depends on the people who ask it. From the geographer's point of view, a visualization

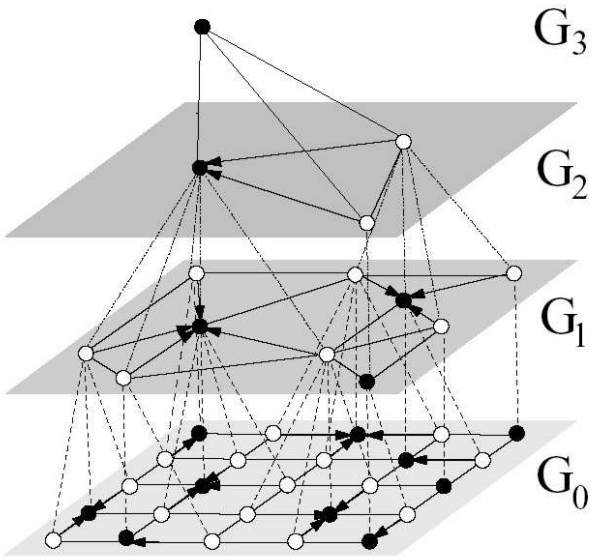


Figure 1. A small graph pyramid based on a 5×5 pixel image (taken from [6]).

tool improves among other things the better understanding of

- the contraction rule's effects on the pyramid structure,
- the verification of the contraction rule set and definitions,
- the correlation between pyramid levels and thematical maps, and
- the role of node and edge attributes.

Interaction and an appropriate visualization of information are important aspects. The visualization of the whole graph pyramid is well suited for navigation. All needed information is stored in this data structure and there is a clear and logical connection between their components. For example, the user can select a special pyramid level and the visualization system opens a view with the represented thematical map computed by a down projection to the base level, see Section 3.4. So, the user can easily develop new contraction rules and verify them with the help of the visualization. He/she can analyze the use and computation of contraction kernels level by level as well as the resulting thematical maps. Here, the system should display the correlation between pixels/pixel areas in the thematical map and the corresponding node in the distinguished level. There are similar scenarios with respect to the other points summarized in the list above.

Another group of users are researchers in Pattern Recognition and Image Processing. For example, these people use

graph pyramids built by DGC for partitioning images [7]. A visualization leads to the same advantages as decribed before, but additionally it supports studies about the structure of a pyramid itself, like its node distributions or height in relation of a specific input image. Thus, the researcher can debug algorithms that perform the DGC process and ask for statistical information (e.g. the node reduction factor). Another point of interest are comparisons of two or more (satellite) images by means of their graph pyramids.

3. Visualization Model and Views

Our visualization approach was implemented in Java [2] and Java 3D [12, 15]. It supports currently four views (2D and 3D) on the pyramid structure as well as related data. The *Graph Pyramid View* can be used for common navigation purposes since graph pyramids are the central structures that store all information. Furthermore, our approach supports the visualization of level graphs and subtrees specified by contraction kernels. Attributes and some statistical information are displayed within the *Attribute View* and within the *Statistics View*, respectively. The Graph Pyramid View and the Statistics View consist of several parts, i.e., closely related subviews in order to display different visualization needs. The fourth *Thematical Map View* is used to show the thematical maps for a selected level. All views are connected with an own event management system (see Section 5) which assumes responsibility for a correct interaction behaviour, e.g. a selection of a node in the Graph Pyramid View leads to a selection of the same node in all subviews or to a focusing on the node's attributes. We could also load several images and represent four views for each generated graph pyramid.

3.1. Graph Pyramid View

This central visualization presents three different subviews closely connected with each other. Note that the screenshot examples discussed in the next sections only show visualizations based on very small input data.

Pyramid Subview The Pyramid Subview gives the full view of the data structure, i.e., each plane graph is drawn level by level. The graph nodes are visualized by spheres while the edges are thin cylinders, each connecting two spheres. Figure 2 shows the graph pyramid of a small two-colored image. Light green lines which connect two graphs of different hierarchies represent the contraction kernels. The color of an image pixel is an important attribute, which exists for every node. Therefore, the spheres get their colors from their according color attributes which are either red or blue in our example.

While Figure 2 only shows a small image, bigger images produce bigger graph pyramids with more visualization objects. Standard mouse-based navigation techniques, like rotating, panning, or zooming, are supported. In practice, this visualization of a complete graph pyramid could be too complex for users to get deeper information. Instead, a visualization of separate levels of the graph pyramid is more desirable, where each level can selectively be shown or hidden. This feature is also supported by our prototype implementation.

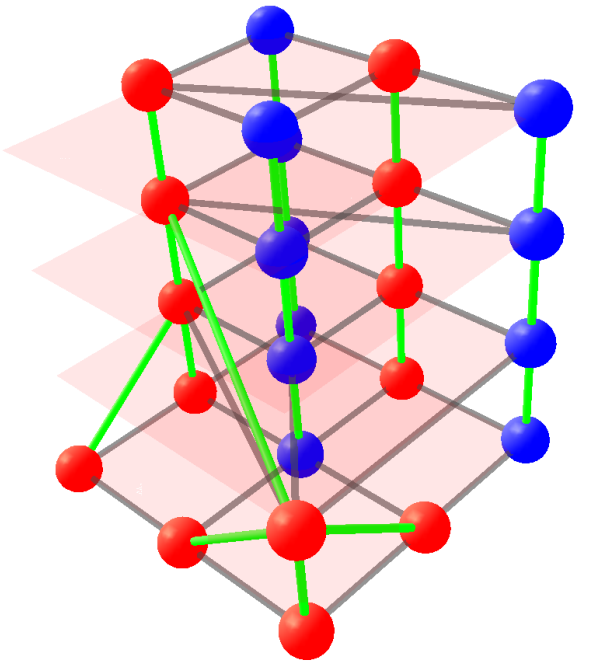


Figure 2. The graph pyramid of a 3×3 pixel image.

Problems arise in the way transparent objects are drawn in Java 3D: If you carefully look at Figure 2, you can see that some slightly transparent edges are painted in front of contraction kernels. Actually, they should be behind them. This is a known issue with Java 3D and the only way to completely avoid it is to not use transparent objects at all. However, this is not really a huge problem because it does not catch the user's eye.

Level Subview This part of the Graph Pyramid View shows the actual selected level also as 3D visualization. There are the same navigation facilities as in the Pyramid Subview. From the user's point of view, the visual comparison between the level graph and the thematical map visualization (see below) is very important. Thus, he/she can see the mapping of the graph nodes and the appropriate image region of the thematical map. It is possible to completely hide this view.

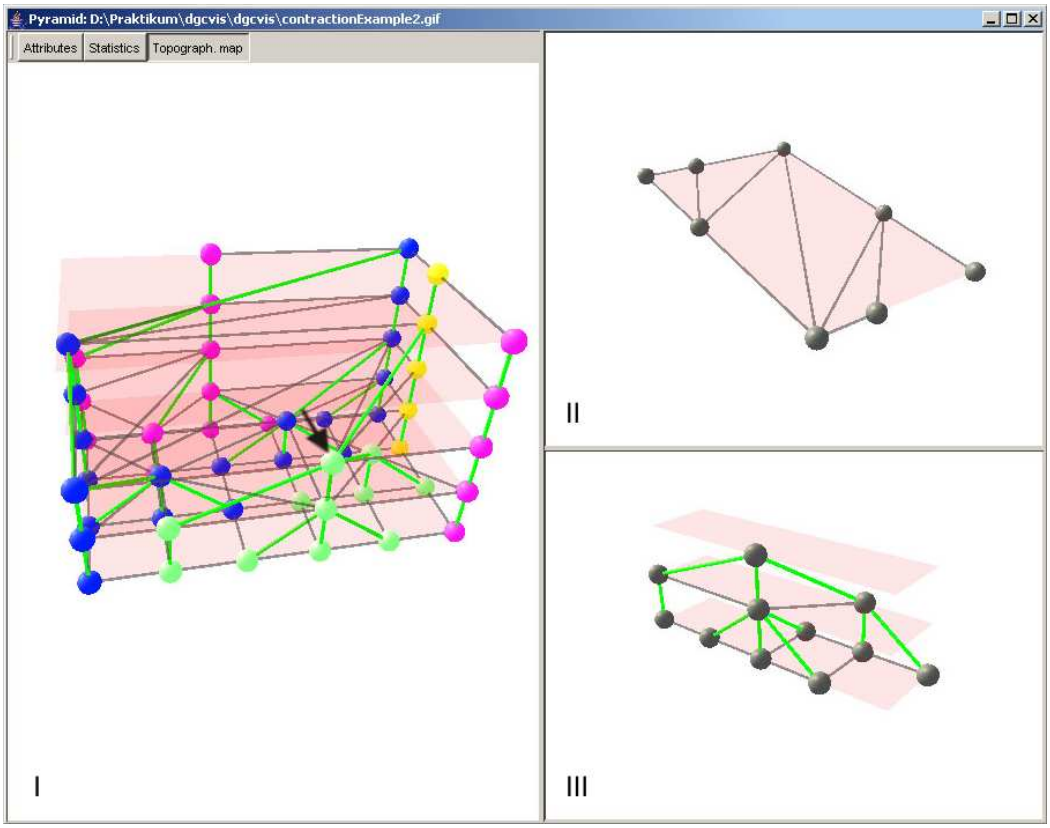


Figure 3. Pyramid (I), Level (II), and Subtree (III) Subviews.

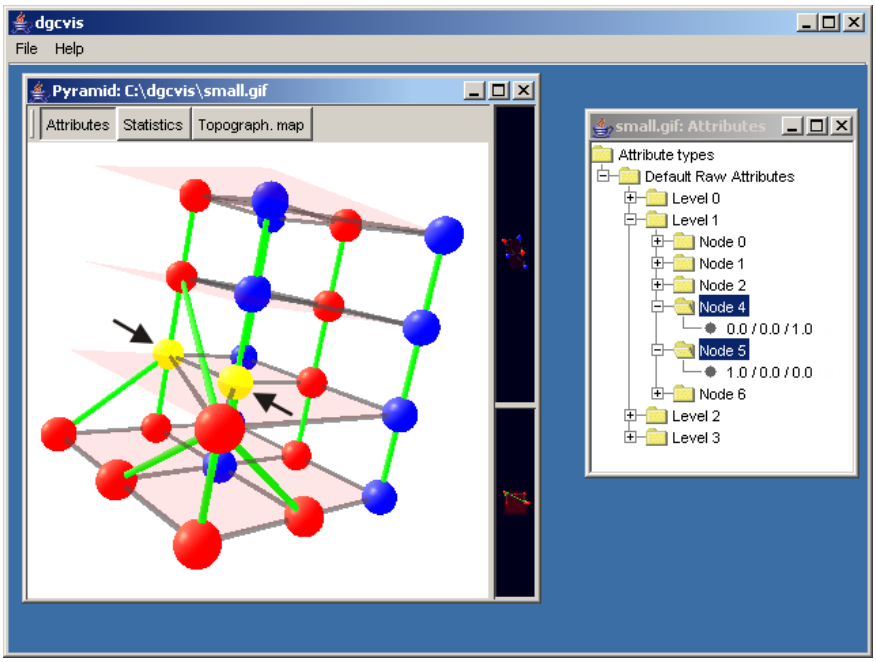


Figure 4. Attribute View (window on the right) with two selected nodes.

Subtree Subview After selecting a node in the Pyramid Subview, the connected subtree is computed and placed in this view. The subtree computation starts with the selected node and is linked via contraction kernels to the lower levels. These connections are traced and the resulting tree is copied into a new pyramid. In this way, the presentation of the whole data structure is reduced to the level, subtree, and node of interest. It is also possible to completely hide this view.

Figure 3 shows all three subviews of the Graph Pyramid View. The black arrow in the Pyramid Subview symbolizes a specific node selection at the third pyramid level. At this moment, the selected node's subtree is computed by the DGClib. The subtree is highlighted in the Pyramid Subview and additionally displayed in the Subtree Subview. The third level is shown in the Level Subview.

In future versions of our application, using this view could be one way to create contractions from the view to define new contraction rules. By selecting some nodes, the visualization tool can compare their attributes and then create some contraction rules providing a contraction of the selected nodes for the user. In this way, it would be possible to create a "What If" pyramid with the possibility to check own hypotheses.

3.2. Attribute View

Visualization of attributes of a pyramid's nodes and edges is done by two means: Enumeration of nodes and edges by attribute list types and level, and statistical distribution of attribute list types. The second will be described later.

In the Attribute View, users can hierarchically navigate through a so-called "TreeView" (see [14]) with all the attribute list types existing in the graph pyramid. Those attribute list types are grouped by level and then in the next hierarchy level by node or edge. Figure 4 shows both, a Pyramid View of our example graph pyramid to the left and the Attribute View to the right. Node 4 and 5 of level 1 are selected in the Attribute View. In the Pyramid Subview those two nodes got highlighted (see black arrows). The values of both nodes' attribute list type can be seen in the deepest TreeView hierarchy level. *Default Raw Attributes* is the name of the attribute list type which specifies the color of a node. This attribute list type has three elements: the RGB values of the color. Here, node 4 is colored blue and node 5 red.

Right now, we primarily work only with the color attribute because there are not any appropriate input examples yet. However, the system can work with an unlimited number of different attributes, such as humidity, population density, vegetation, etc.

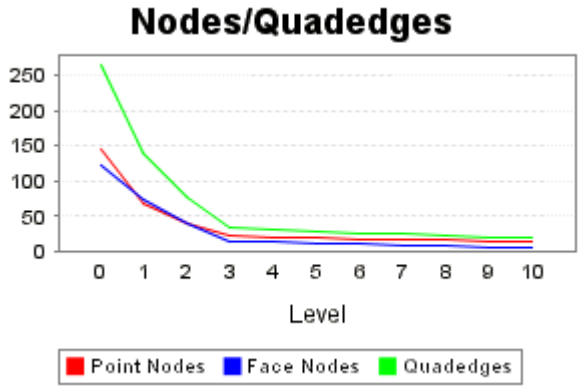


Figure 5. Statistical information of distribution of point nodes, face nodes and quadedges over levels.

3.3. Statistics View

Global statistics are shown as chart diagrams within a separate window. At present, the statistical information a user can obtain are distributions of

- point nodes, face nodes, and quadedges [5],
- attribute list types of point nodes,
- attribute list types of face nodes,
- attribute list types of quadedges,
- contraction function sets in point graphs, and
- contraction function sets in face graphs,

each over levels. Figure 5 shows a distribution chart of point nodes, face nodes, and quadedges, which gives an understanding of the reduction factor of the graph pyramid. The (relative) number of attributes per level are useful for recognizing which attributes get lost during contractions (this can happen if they appear to be irrelevant at a certain level of abstraction).

This screenshot example mainly refers to the needs of users from the Pattern Recognition and Image Processing community. But it is also possible to show statistical information about attributes (see points 2-4) based on geographical input data.

3.4. Thematical Map View

A 2D approach for visualization is the Thematical Map View (TMV). This view shows for a selected level, similar to the Level Subview, its projection to the input image (called *down projection*) on which the graph pyramid is based. Among other things, attributes of each node at the level (for example the RGB value) are forwarded to nodes

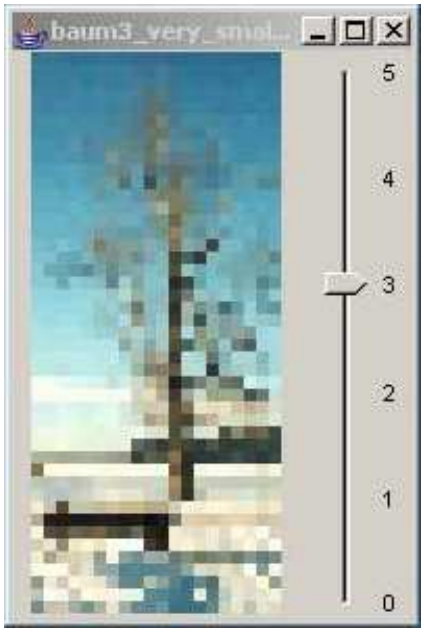


Figure 6. Thematical Map View

of the base level along the tree paths built from the contraction kernels. In case of GML graphs as input, the down projection is performed with the help of a path per node for the specification of the area shape that is represented by the node. This information is part of the GML specification.

The view contains a slider to change the selected level, see Figure 6. If there are selected nodes in the Pyramid View, then the TMV displays the down projection only on the basis of the selected nodes at the selected level of the TMV. If there is a whole subtree selection in the Pyramid View (cp. Section 4), then just the nodes in the selected level are shown. The reason for this approach is that the projections would overlap if the TMV would map all selected nodes of the subtree.

4. Navigation

A key point for better understanding a graph pyramid is the possibility of navigating through the structure. The visualization of the pyramid can be seen as one piece that can be grasped and manipulated. Users can look at the pyramid from every possible viewpoint. They can move (pan) the pyramid, rotate it and zoom in and out of it using the mouse. We decided not to use keyboard interaction but only mouse input for handling the pyramid, as a mouse—especially when using dragging—gives the more natural feeling for manipulating objects. The Java 3D API already features classes that assist us in grabbing, moving, and zooming visual objects.

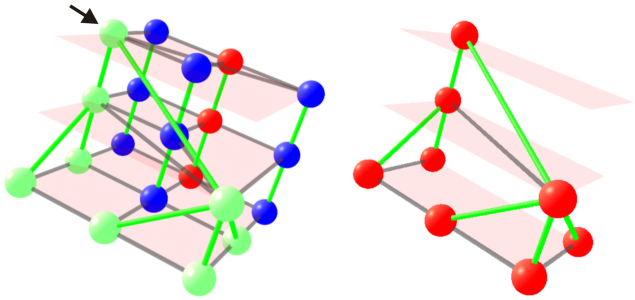


Figure 7. The left picture shows a selection of one node. When the topmost green node is clicked (see black arrow), all of its subnodes are automatically selected as well. At the same time the selected subpyramid is drawn, which can be seen in the right picture.

In general, there are two different node selection types: attributes selection (which we have described in Section 3.2) and subtree selection. Both selection types exist side by side. Selected nodes get a different color. We defined subtree selected nodes to be light green and attributes selected nodes to be yellow. These default values can be changed by the user. If a node is both subtree and attributes selected then its color will be a mixture of both. By clicking on a node, a subtree selection can be made. This shows Figure 7. There, one node was selected by mouse which results in eight highlighted nodes altogether. The full subpyramid can be seen on the right.

The Graph Pyramid View that was described in Section 3.1 represents a so-called Focus&Context technique, see the article of Herman et al. [8] or the anthology of Card et al. [1]. The user can focus on a specific level or subtree whereas the context of this focus is displayed in the Pyramid Subview.

5. Implementation

At the beginning of the planning phase of the application some points became clear: The data structure will typically be huge, computing the data structure will take a lot of time and there is not enough place on a monitor to visualize all of the data. These problems could be partially solved by skillful implementation techniques and view design.

5.1. Interface to DGClib

At first, we had to handle the problem that the computational part of our application takes place within the DGClib

which was implemented in C++ for performance reasons. There are several possibilities to transfer the data structure together with all references to our Java application for visualization. Due to the fact that platform independence was a requirement, we implemented two corresponding interfaces:

Get Data from JNI One solution was to compile the framework as dynamic library and import it to Java via JNI (Java Native Interface). So, it was possible to keep the major computation load in C++ and simply use the results in our Java program. By compiling the DGClib for different platforms, it is available on different operating systems. Future releases may also have the possibility to use different libraries at the same time to be able to compare the changes made.

Get Data from Network Interface Another solution would be to develop a distributed application to acquire data from a central server having a higher processing power than the client computer on which our Java application runs. The modular structure of the data structure is able to support this solution very well. For the communication it is necessary to implement a server interface as servlet and let the application communicate via RMI (Remote Method Invocation) or with a simple protocol designed for this purpose.

5.2. Architecture

To get a clear distinction between the data structure and the Java-based visualization, the consequent implementation of a MVC (Model, View, Controller) model [4] was forced. As a consequence, there are no direct references between them.

Model The computation of the graph pyramid and related information is performed by the DGClib using dual graph contraction algorithms. Details can be found in several articles [7, 10].

Controller The communication between graphical objects is based on an event system that is very similar to the event system used by Java. A Java class `EventControl` is used for registration of listeners and distribution of events. The used types of events are:

- `ControlEvents`: These are generated by creation, activation, or removal of windows.
- `DataEvents`: They inform about all changes in association to the data structure, like loading of levels, changes in nodes, edges, or contraction kernels.
- `GUIEvents`: They are used for the communication with the views. User interactions (especially highlights) are distributed by this event type.

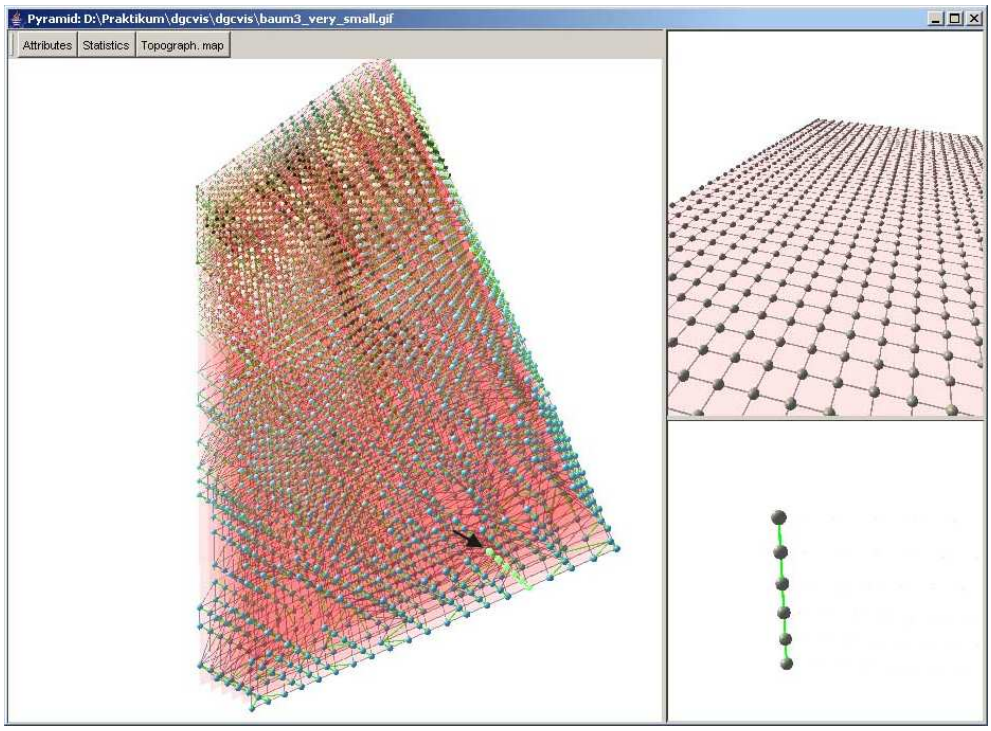
The controller component guarantees a consistent and stable interaction behaviour, easy extensibility with new features, and a coherent coordination of our views. For example, the node selection mechanism is realized by sending a parametrized `GUIEvent` object. After the selection of a node within any view, e.g., within the Pyramid View, this event is forwarded to all other coordinated views. The node can be deselected in any other view. In general, a clever implementation of the controller amortizes the effort relatively quickly.

View The huge amount of objects (on average about the number of pixels times 30 for an image as input) implies the need for fast rendering. But we also intended a high-level abstraction mechanism to specify 3D visualizations. Thus, we decided to use Java 3D, which is based on OpenGL [13]. The objects are added to a scene graph (the universe) and rendered with hardware support. This solution has some special requirements on the operating system that provides the handling of the drawing areas. This implies some problems with frame visibility, because Java draws the frames itself and it seems to be not fully compatible with the OS. It occurs that hardware supported areas always overlap frames painted by Java, independent if they are in front or not. This problem should be solved with the next generation of Java 3D.

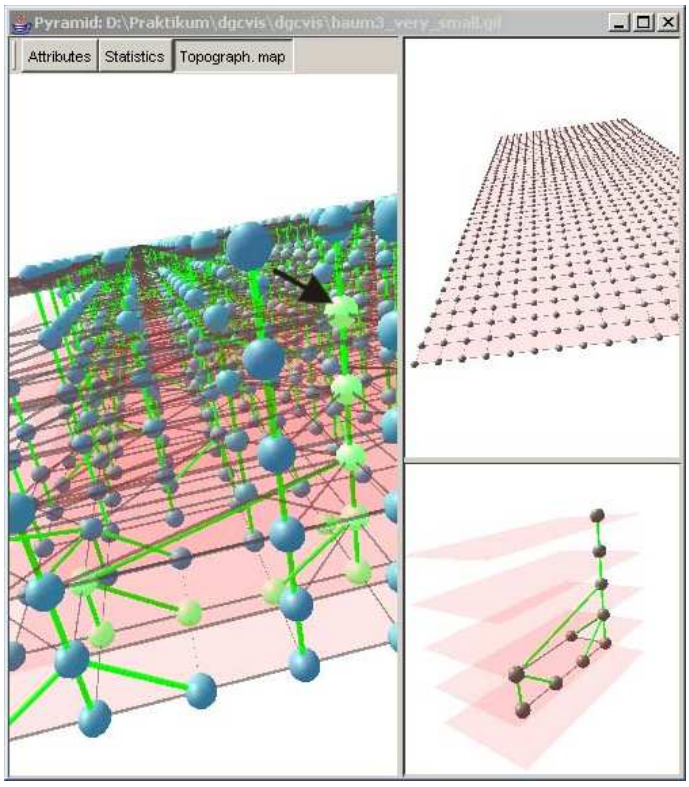
5.3. Performance Analysis

Due to the huge size of the data structure it is even with small images obvious that a visualization of the whole graph pyramid will lead to difficulties in memory usage and visualization space. Up to a specific limit, these problems can mostly be solved with clever solutions which focus on special visualization aspects (e.g. TMV, Subtree Subview, Level Subview, ...). At the moment, the end of the technical possibilities is reached soon. Currently, our approach works fine for images up to approximately 50×50 pixel. A PC used for development (AthlonXP 2800+, 1024MB Ram, GF4, WinXP) requires for the computation of a 100×100 pixel image by the DGClib about 30 seconds. The resulting graph pyramid was built with 20 levels containing in total 35.000 nodes, about 50.000 edges and some 1.000 contraction kernels. The creation of the scene graph with Java 3D took about 1 minute. The memory peak was about 900MB and the rendered scene graph took about 200MB permanently. The speed of rendering was very low and just reached 2-3 fps.

Fortunately, we can avoid most of the above problems by using pre-segmented input graphs via the GML exchange format, cp. Section 2. In this way, the lower levels of the graph pyramid can usually be discarded, because they are often not so important for users like geographers or landscape ecologists. The number of graphical objects can be drastically reduced and as result the visualization works fine with larger input images.

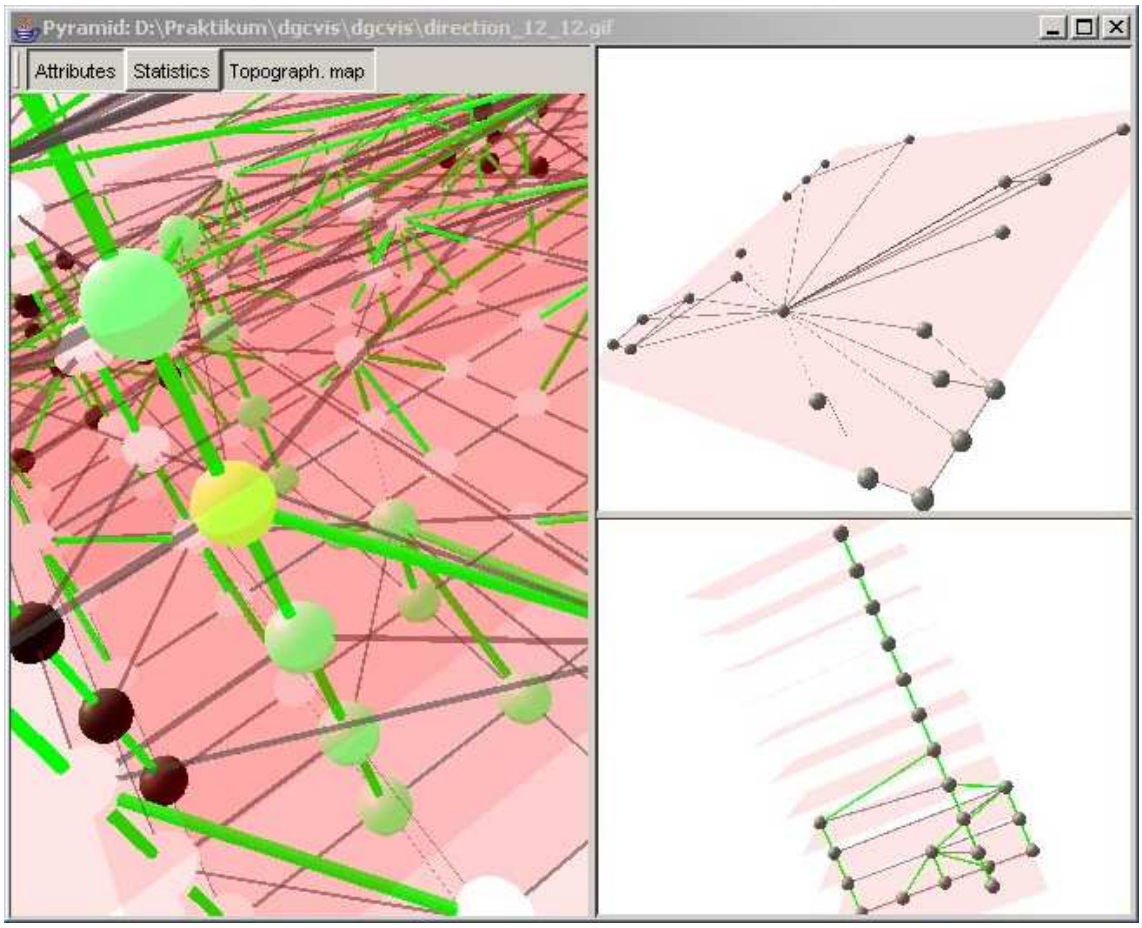


(a) Graph Pyramid View with a subtree selection.

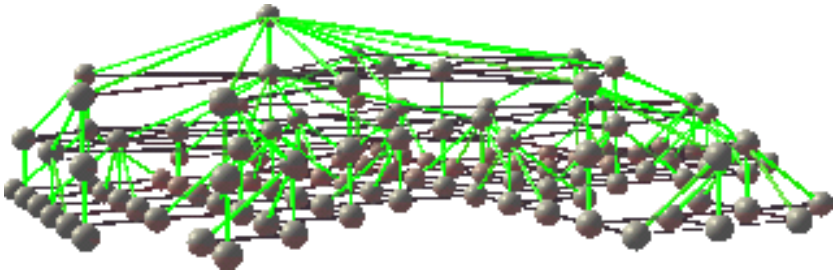


(b) Graph Pyramid View with another subtree selection.

Figure 8. Graph Pyramid View of the 20×45 pixel color image example.



(a) Graph Pyramid View

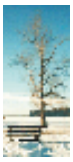


(b) A complex subtree extracted from the Subtree Subview.

Figure 9. Graph Pyramid View of the 12×12 pixel B/W image example.

6. Application Examples

The input of the screenshot examples in Figure 8 is a small image (a tree and a lawn seat in front of a cloudy sky at the background, see Figure 10(a)) of size 20×45 pixels and with 256 colors. The Pyramid Subview on the left shows a very compact graph pyramid of height 6. Its reduction factor from level to level is very inconspicuous. The same is illustrated by the Level Subview right above. Regarding the Subtree Subview of Figure 8(a), we see that the selected subtree forms a simple path and not a real tree with a higher node degree. It seems that the implementation of the DGC algorithm together with the current contraction rule set has difficulties with this kind of input. But there are some contractions which can be observed in our Subtree Subview in Figure 8(b).



(a)

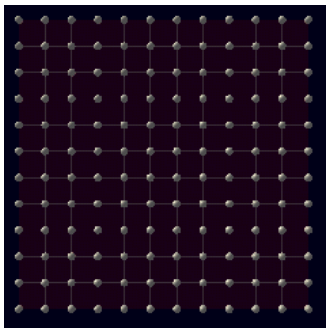
(b)

Figure 10. Input examples

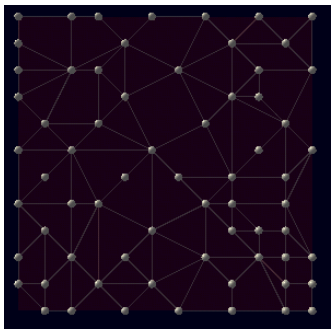
Input of our second screenshot example in Figure 9 is a smaller B/W image (four numbers on a white background, see Figure 10(b)) of size 12×12 pixels. The Pyramid Subview shows a graph pyramid of height 10. Its reduction factor from level to level is better as in our first example. The Level Subview shows the graph at level 4 with only 22 nodes. Regarding the Subtree Subview of Figure 9(a), we currently see that the selected subtree forms a small tree. In Figure 9(b) a bigger subtree is displayed. Here, the user can study the contraction process together with the Level Subviews shown in Figure 11 very well. The graph of the base level in Figure 11(a) forms a regular grid as described in Section 2. Then, nodes and edges are reduced by the DGC algorithm level by level. For example, the center node of level 4 in Figure 11(d) represents the white background of the input image.

7. Conclusions and Future Work

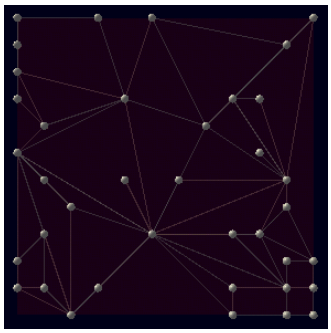
In this paper, a novel approach for the visualization of hierarchies of plane graphs, called graph pyramids, was presented. They are used for collecting, storing and analyzing geographical information based on images or other input data. The visualization covers several visualization needs of geographers and of reseachers working on pattern recogni-



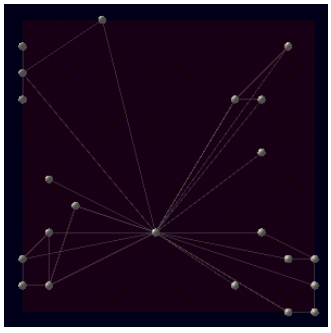
(a) Level 1



(b) Level 2



(c) Level 3



(d) Level 4

Figure 11. Level Subviews of the lower four pyramid levels.

tion algorithms. As far as we know, our visualization tool is the first one in the area of interactive visualization of graph pyramids. All implemented views are closely related. They offer many interaction and exploration possibilites to discover new correlations between contraction rules, thematical maps, and the graph pyramid's structure. We have implemented a prototype version of the visualization in coorperation with our project partners the Institute of Surveying, Remote Sensing and Land Information at the BOKU Vienna as well as the Pattern Recognition and Image Processing Group at the Vienna University of Technology.

We plan to evaluate our visualization tool together with our users in the near future in order to improve its applicability and usefulness. There are a lot of challenging problems in this area that we want to solve, e.g., visualizing the evolution of landscape changes, interactive input of contraction rules from the visualization (cp. Section 3.1), visualization of graph pyramid changes through movements of objects within an image sequence, and the visualization of pyramid comparisons. From the viewpoint of Information Visualization, we have to find better navigation solutions and an advanced Focus&Context approach. This will be one of our main research aims because an extensive study of the existing literature yielded no applicable results for this kind of problems.

Acknowledgements

We would like to thank Michael Schreyer for implementing the DGClib interface and Günther Raidl for carefully proof-reading this paper. Furthermore, we wish to thank the members of the Institute of Surveying, Remote Sensing and Land Information at the BOKU Vienna as well as the members of the Pattern Recognition and Image Processing Group (PRIP) at the Vienna University of Technology for their useful ideas and support.

References

[1] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[2] D. Flanagan. *Java in a Nutshell*. O'Reilly, 3. edition, 2000.

[3] K. S. Fu and J. K. Mui. A Survey on Image Segmentation. *Pattern Recognition*, 13(1):3–16, 1981.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Massachusetts, USA, 1995.

[5] L. Guibas and J. Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[6] Y. Haxhimusa, R. Glanz, M. Saib, G. Langs, and W. G. Kropatsch. Logarithmic Tapering Graph Paramid. In L. V. Gool, editor, *Proceedings of the 24th DAGM Symposium 2002*, LNCS 2449, pages 117–124, Zürich, Swiss, 2002. Springer.

[7] Y. Haxhimusa and W. G. Kropatsch. Hierarchical Image Partitioning with Dual Graph Contraction. In B. Michaelis and G. Krell, editors, *Proceedings of the 25th DAGM Symposium 2003*, pages 338–345, Magdeburg, Germany, 2003. Springer.

[8] I. Herman, G. Melançon, and M. S. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[9] M. Himsolt. GML – Graph Modelling Language. `www.uni-passau.de/Graphlet/GML`, 1997.

[10] W. G. Kropatsch. Building Irregular Pyramids by Dual Graph Contraction. *IEE-Proc. Vision, Image and Signal Processing*, 142(6):366–374, 1995.

[11] N. R. Pal and S. K. Pal. A Review on Image Segmentation Techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[12] D. Selman. *Java 3D Programming*. Manning, 2002.

[13] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4*. Addison Wesley, 4. edition, 2003.

[14] K. Walrath and M. Campione. *The JFC Swing Tutorial: A Guide to Constructing GUIs*. Addison Wesley, 1999.

[15] A. E. Walsh and D. Gehringer. *Java 3D API Jump Start*. Prentice Hall PTR, 2002.