

OBJECT-ORIENTED MODELING FOR THE EXTRACTION OF GEOMETRY, TEXTURE AND REFLECTANCE FROM DIGITAL IMAGES

Hardo Müller
Institut für Photogrammetrie
Universität Bonn
Nußallee 15, 53115 Bonn, Germany
Ph.: +49-228-73-2721, Fax: +49-228-73-2712
e-mail: hardo@ipb.uni-bonn.de

KEY WORDS: Semi-Automatic Extraction System, CORBA, JavaBeans, GIS, CSG

ABSTRACT

A semi-automatic system for extracting topographic features is being migrated to an object-oriented design for better maintainability. For that purpose an object-oriented model of the extracted objects and the extraction methods is required. Moreover an appropriate software component model for the exchange with other systems is needed. We have modeled a class hierarchy for objects, that can be semi-automatically extracted from digital images. These objects are characterized by geometric, textural and reflectance properties. We have classified the extraction methods and modeled the message transfer of an interactive extraction method. The component technologies CORBA and JavaBeans™ were used to make the extracted objects and system components available for other systems. We found out that an access to the objects of the Semi-Automatic System by operation calls allows a more flexible data transfer and control of the system than standard file transfer. Therefore Geographic Information Systems (GIS) should support appropriate software component models to co-operate optimally with interactive or semi-automatic feature extraction systems.

1 INTRODUCTION

A semi-automatic system for the extraction of buildings from digital images has been developed at the Institute of Photogrammetry, University of Bonn since 1993. The operator uses the system in a mono-scopie mode supported by interactive and automatic extraction methods. This system has been presented in several earlier publications (Lang and Schickler, 1993, Lang and Förstner, 1996, Englert and Gülch, 1996, Gülch and Müller, 1997, Müller, 1997, Müller, 1998).

The originally procedural structured software has been migrated to an object-oriented design. Thereby we have obtained an essentially higher maintainability of the software. In contrast to simple data structures we are now able to produce objects as output. We get a substantially higher level of information from these objects, because the informations are transferred by operation calls, which is not possible with the classical method of reading data from a stream.

For the object-oriented system design we need an object-oriented model of the extracted objects and the extraction methods. Objects, extracted by the semi-automatic system, are characterized by geometric and textural information. Texture, extracted from digital images, depends on lighting conditions, camera properties and image processing influences. For a more object-specific description we need to model the reflectance properties of an object. Extraction methods as well as external components have requirements on the object-oriented model of the extracted objects. E. g. most of the extraction methods need a parametric description instead of a polyhedral one and some Virtual Reality Modeling Language (VRML) browsers can handle only rectified texture. The exchange of messages between method-specific objects and platform-specific objects is of importance for the object-oriented model of the extraction methods. This concerns especially interactive methods. For flexible applicability we need a clear separation of these two object groups. Moreover we need techniques to pass the extracted objects to external components. Thereby we have to consider existing standards. The object-oriented model of the system has to be kept flexible, that an adaption to future standards can be easily performed. In this context the problem of object-oriented modeling is on the one hand to model the natural properties of the objects as real as possible and on the other hand to consider

implementation aspects and platform independence for a flexible reuse.

This paper presents the object-oriented model of the extracted objects, which are characterized by geometry, texture and reflectance properties (cf. section 2). For a graphic representation of our dynamic and static object-model, we use the Unified Modeling Language (UML), which is described in (Rat, 1997) and has adopted elements of the OMT (Rumbaugh et al., 1991) and the Booch-method (Booch, 1994). A classification of the extraction methods and an object-oriented model of the message transfer in an interactive method is depicted in section 3. The application of the object-transfer techniques CORBA, Java™ Object Serialization and JavaBeans™ for semi-automatic object extraction is described in section 4. In section 5 some examples are presented. Finally, we discuss in section 6 the characteristics, which a GIS should have, in order to co-operate with semi-automatic object extraction.

2 EXTRACTED OBJECTS

The objects, which are extracted from digital images, are attached with geometric- and textural information. The textural information is taken from the reflectance properties of the material. Our first task is to design a class structure, which represents an appropriate description of the geometric, textural and reflectance information.

2.1 Geometry

Points, Edges and Faces The basic primitives of our geometric description are points, edges and faces. So we model a class structure, consisting of these primitives. This canonical model links points, edges and faces by pairs with bidirectional associations. Figure 1 shows a class diagram of this most intuitive way designing geometric primitives.

Polyhedral Models The above mentioned primitives can be composed to simple polyhedral models. With polyhedral models we are able to describe the geometry of topographic objects like buildings. The class structure for a polyhedral model is shown in figure 2.

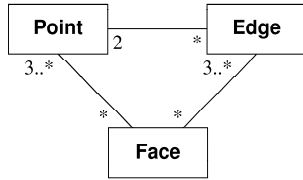


Figure 1: A simple model for geometric primitives like points, edges and faces. The association lines between the classes represent bidirectional neighborhood relations. They are attached with multiplicities at their end (Numbers or *-symbol for N -times multiplicity).

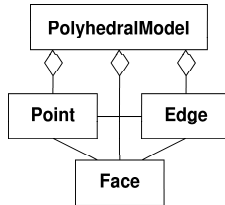


Figure 2: Class structure for a simple polyhedral model. We model a *PolyhedralModel*-class as a composition of *Point*-, *Edge*- and *Face*-objects, whereas the operations of the *PolyhedralModel*-class are responsible for the generation of *Edge*-objects from given faces and points. The diamond symbols represent aggregation associations. Multiplicity is omitted for clearness.

Most of CAD and virtual reality applications use polyhedral models for geometric description. For the interactive extraction of geometric objects from digital images, the use of simple polyhedral models is disadvantageous. The problem is the large number of parameters, which are needed to describe one model. E. g. a polyhedral model of a saddleback building consists of 10 3D-Points, 15 Edges and 7 Faces. This means, that alone 30 Parameters are needed to describe the geometry of the model in a polyhedral mode (cf. Figure 3).

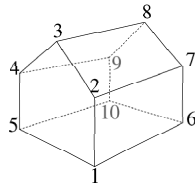


Figure 3: Saddleback building as a polyhedral model. Since each of the 10 corner points is specified by 3 coordinate values, 30 parameters are necessary to describe the geometry in this representation. Parameterization therefore is useful.

Parametric Models For the semi-automatic extraction, we need a model, which gets along with fewer parameters. A parametric model fulfills this request. These models can effectively be used for buildings with flat, saddleback or hip roof, requiring 3, 4 or 5 parameters for specifying the form. The coordinates of the corner points are calculated by a design matrix, which is given for each model type. This process is described in more detail in (Lang and Förstner, 1996) and (Müller, 1998). The class structure of these classes is shown in figure 4. Every modification of the model parameters causes an update of the point coordinates.

CSG models Complex buildings are not describable by a few parameters. Therefore we use the Constructive Solid Geometry (CSG) principle for a combination of parametric models (Hoffmann, 1989). The CSG model is a tree structure, which consists of a combination of solid primitives by the logical operations union, intersection and difference (Englert and Gülch, 1996, Müller, 1998). Fig-

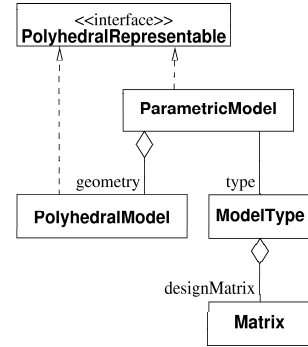


Figure 4: Class structure for a parametric model. The *ParametricModel*-class contains a *geometry*-object, which is an instance of the *PolyhedralModel*-class, and it is associated to a *type*-object, which is an instance of the *ModelType*-class. The operations of the *ParametricModel*-class are responsible for the state of the *geometry*-object, which must be consistent with actual parameter values. The point coordinates and topological structure is calculated by the operations of the *ModelType*-class, which uses the *designMatrix*-object in the manner as described in Lang and Förstner, 1996. Since parametric models can be represented in a polyhedral manner, we have designed a *PolyhedralRepresentable*-interface, which is implemented by the *PolyhedralModel*- and *ParametricModel*-class.

ure 5 depicts our class-structure for CSG models. This class structure is designed similar to the Composite design pattern, which is intended to model tree-structures (Gamma et al., 1995).

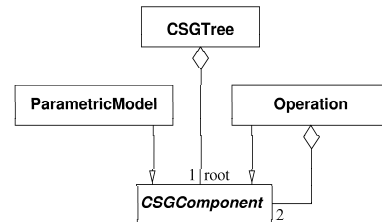


Figure 5: Class structure of a CSG tree. The *CSGComponent*-class, which is the base class of the *ParametricModel*-class and *Operation*-class, represents the node of a CSG tree. The root component of the tree is a part of the *CSGTree*-class.

2.2 Texture

We also extract textural information. This information is needed e. g. in virtual reality applications for a more realistic view. For this purpose we have to identify common representations for textural information. VRML-files e. g. need the model points to be declared in texture coordinates and the corresponding images for the texture data (VRM, 1997). Texture coordinates are 2D-coordinates related to the texture data. Figure 6 shows, how the texture coordinates of a face are related to the object space.

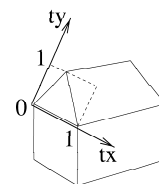


Figure 6: Texture coordinates (tx, ty) on a roof face of a hip roof building. This relation is represented by a homogeneous 4×4 matrix.

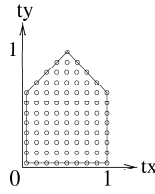


Figure 7: Raster of texture elements on a front face of a saddleback-roof.

After the raster of texture elements is specified as shown in figure 7, the texture extraction process can start. For each texture element, the texture coordinate is converted first to an object coordinate and second to an image coordinate using the orientation of the image. From this image position the color or grey value is taken. We assign this rectification process to the extraction of texture, because not all visualization tools can handle distorted texture data.

Since we want to extract texture from several images, we need a criterion to decide, which of the images shall be taken for texture extraction. This is done by the calculation of a weight for each texture element. The weight becomes higher, if the area of the projected texture element in the image increases and the angle α between view direction and surface normal decreases. The weight is calculated as following:

$$w = \begin{cases} A \left(1 - \frac{2\alpha}{\pi}\right)^k & : \alpha < \frac{\pi}{2} \\ 0 & : \alpha \geq \frac{\pi}{2} \end{cases}$$

where

- w = weight,
- A = area of the projected texture element,
- α = angle between surface normal and view direction,
- k = factor of α influence.

For preferring a perpendicular view set k high.

If another image is extracted, the weight is compared with the previously calculated weight, and the texture element gets the data from the new image, if the weight is higher.

The class structure of a face with mapped texture is shown in figure 8. This design enables a straightforward access to rectified texture for external objects, like a VRML file generators, visualization tools etc.

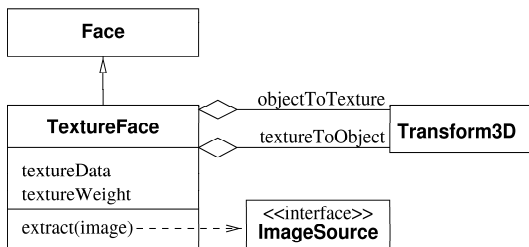


Figure 8: Class structure of a face attached with texture. The textural information is modeled by a *TextureFace*-class, which is inherited from the *Face*-class. The *TextureFace*-class contains *Transform3D*-objects which perform the conversion between object coordinates and texture coordinates by an affine transformation. For the texture extraction an oriented image is needed, which implements the *ImageSource*-interface. Thus, the *extract*-operation uses the *ImageSource*-interface.

2.3 Reflectance

To get object-specific properties from the image data, we have to model the reflection behavior of surfaces. In general the reflectance of a surface can be expressed by the Bidirectional Reflectance Distribution Function (BRDF):

$$f_r(x, \Theta_i, \Theta_o) = \frac{dL_o(x, \Theta_o)}{L_i(x, \Theta_i) \cos \vartheta_i d\omega_i}$$

where x is the position on the surface, $\Theta_i = (\vartheta_i, \varphi_i)$ is the direction from which the light comes in, $\Theta_o = (\vartheta_o, \varphi_o)$ is the direction in which the light is reflected and measured, $dL_o(x, \Theta_o)$ is the differential amount of radiance that is reflected at point x in the outgoing direction Θ_o , $L_i(x, \Theta_i)$ is the amount of radiance coming in at point x along direction Θ_i through $d\omega_i$, $\cos \vartheta_i$ is the cosine of the angle between Θ_i and the surface normal at point x , and $d\omega_i$ is a differential angle around Θ_i .

The BRDF is the ratio of the radiance in the direction of the reflected beam to the irradiance caused by the entrance beam. Depending on the physical model there are several possibilities to express the BRDF. For example the Lambertian BRDF for diffuse reflection

$$f_r(x, \Theta_i, \Theta_o) = \frac{k_d}{\pi}$$

or the modified Phong model with a diffuse and a specular part (Lafortune and Willems, 1994)

$$f_r(x, \Theta_i, \Theta_o) = k_d \frac{1}{\pi} + k_s \frac{n+2}{2\pi} \cos^n \alpha$$

or the BRDF in terms of surface scattering modes (Koenderink and vanDoorn, 1996). This model allows to represent parametrically a complex BRDF.

$$f_r(x, \Theta_i, \Theta_o) = \sum_{nml} a_{nml} S_{nm}^l(\vartheta_i, \vartheta_o, |\varphi_i - \varphi_o|)$$

where α is the angle between perfect specular reflective direction and the outgoing direction, k_d is the diffuse reflectivity, k_s is the specular reflectivity, $S_{nm}^l(\vartheta_i, \vartheta_o, |\varphi_i - \varphi_o|)$ are the basis functions for the surface scattering modes, and a_{nml} are the coefficients for the surface scattering modes.

The class structure for reflectance properties is shown in figure 9. In this design we use the polymorphism principle to express the abstraction of the BRDF. A tool, which extracts reflectance, has to use at least one of the concrete models expressed by the subclasses of the BRDF-class, because abstract classes cannot be instantiated. For this design we need a persistence mechanism, which is able to keep the abstraction principle at the one hand and to restore the BRDF-object with the originally used subclass at the other hand. We will discuss in section 4 which techniques are nowadays available to fulfill this request.

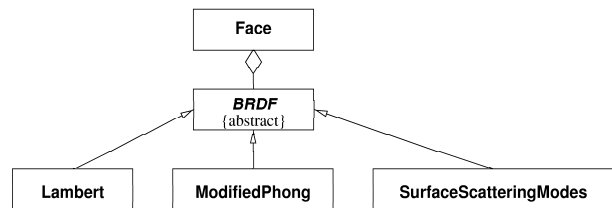


Figure 9: Class structure for the modeling of reflectance properties with several BRDF models. The Bidirectional Reflection Distribution Function is modeled by an abstract class *BRDF* from which the *BRDF* models are inherited. The *BRDF*-class itself is part of the *Face*-class.

Method	Automatic	Interactive	Modifies	Uses	Durable
Model selection		X	Parametric model	1 Image	
Parameter adaption		X	Parametric model	1 – 2 Images, Orientation	X
Ground point measurement	X	X	Parametric model	≥ 2 Images, Orientation	
Gable point measurement	X		Parametric model	≥ 2 Images, Orientation	
RANSAC technique	X		Parametric model	Extracted edges, Orientation	
Fine adjustment	X		Parametric model	Extracted edges, Orientation	
CSG construction		X	CSG tree	1 Image	X
Gluing	X		Parametric model	CSG tree	X
Texture extraction	X		Polyhedral model	≥ 1 Image, Orientation	

Table 1: *Classification of extraction methods. Some methods are durable, which means, that the method manipulates the model at multiple times during a sequence of interactions.*

3 EXTRACTION METHODS

The Semi-Automatic Extraction System contains several extraction methods. For the object-oriented model of the different methods, we need a classification of the significant properties. Table 1 shows the different extraction methods and the properties, which are relevant for the object-oriented model. The methods are interactive, automatic or semi-automatic. They modify different kinds of objects and need different input data. Some methods are durable, which means that they manipulate an object at several times. We have to deal with a more complex message transfer than in case of a simple operation call. The methods are triggered by messages, which they get e. g. from a graphical user interface. The main features of the particular methods are listed below:

Model selection The first interaction step of model extraction is to select one of the parametric model types from a toolbar. This type is set to the parametric model.

Parameter adaption The adaptation of the parameters is done in a sequence of steps, each specifying one or two parameters of the model. The parameters are changed depending on a sequence of two points specified by mouse clicks and in dependence of the type of primitive. This method is characterized by a high number of message transfers between the platform-specific Graphical User Interface (GUI) and the extraction system. We will describe the object-model of this interaction sequence below.

Ground point measurement The operator selects one point on the ground nearby the building, and the absolute height of the ground is computed automatically. For this purpose we use cross-correlation on the grey values of the images with an epipolar search strategy. This method consists of interactive and automatic elements. (Müller, 1997).

Gable point measurement The absolute height of the building is performed by matching a central point on a gable roof in the same manner as the ground point measurement. The difference is, that the selection of the point can be done automatically.

RANSAC technique *RANdom SAmple Consensus* can be used as a powerful technique to determine a best fit of building parameters from a given set of image edges (Fischler and Bolles, 1981). A more detailed description about applying this technique within this system can be found in (Läbe and Gülch, 1998). To perform this method, extracted line segments are needed.

Fine adjustment An automatic fine adjustment by a robust spatial resection, using all line segments in all images provides an optimal fit of the selected model to the image data. As well as for RANSAC extracted line segments are needed. (Lang and Schickler, 1993, Läbe and Ellenbeck, 1996).

CSG construction The CSG tree is composed by an interactive selection of primitives and operations. This method manipulates the topology of a CSG tree.

Gluing of building parts Describing a building by the combination of primitives or combined primitives requires a precise “docking” of the primitives. This docking is supported by matching and gluing facilities. The former allows to match at least two edges of different primitives and the latter matches and glues exactly two faces of different primitives together (Englert and Gülch, 1996). This method uses the previously extracted primitives of a CSG tree. It activates itself during the interactive parameter adaption.

Texture extraction Texture extraction depends not on the parametric description of a model. It can be performed on a polyhedral model. The technique of texture extraction has already been described in section 2.2.

The object-oriented modeling of the interaction process is shown by a sequence diagram in figure 10. We can divide the participating objects into three groups: Platform specific objects, method specific objects and the extracted objects. The separation of platform specific objects simplifies the integration into other systems. Method specific objects contain the actual extraction process. Since they are platform independent, they can be used in several environments. The interactive adaption method needs to modify the GUI at several times. To keep the method implementation platform independent, we use an observer object for message transfer. This object has a platform independent interface and it is implemented with platform specific operations to control the GUI. The extracted objects are designed independent of the extraction method. Therefore we can apply anytime new extraction methods on these objects without redesigning their object-model.

4 OBJECT EXCHANGE

The extracted objects have to be ported to other systems or end-user applications. During this process, the object-oriented model should not get lost. This means that particularly the operations and abstractions, including data hiding, should be preserved. If one simply writes the attributes of an object to a file, the reader of this file does not know anything about the operations of this object. It is not possible to store an abstract class in this manner. We use actually three techniques which help to avoid this problem.

Application Programming Interface (API) The System can be extended or integrated within another application by using its API. The models and system controls are accessible by C++-classes. Application developers can use the class library to extend the system or to integrate it into another application.

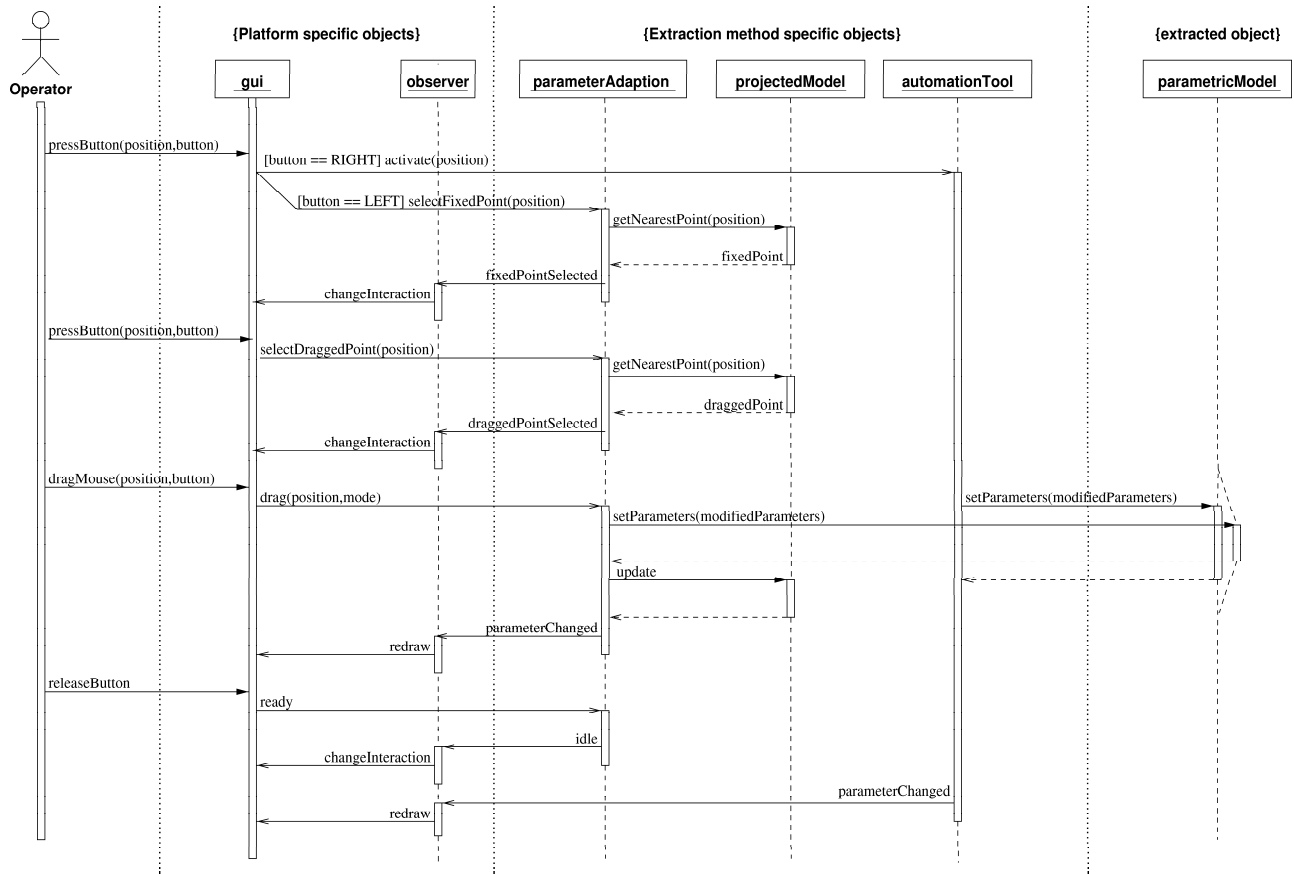


Figure 10: Sequence diagram of the interaction process. This diagram represents a set of messages among objects. The objects are represented by boxes at the top with vertical life-lines below. After the operator has pressed a button, the gui-object decides either to activate an automation tool or to start the interactive parameter adaption. The automationTool-object represents one or more of the automatic methods, which are listed in table 1. It modifies the parametricModel-object and sends a message to the observer-object. This object has a platform independent interface and is responsible for updating platform dependent objects, like the gui-object. In case of interactive parameter adaption the operator selects one corner point to be kept fixed. This is performed by sending a selectFixedPoint-message to the parameterAdaption-object. The parameterAdaption-object gets the nearest point of the cursor position from the projected model and sends the fixedPointSelected-message to the observer-object. The observer-object changes hereon the interaction mode of the gui-object, because the event processing of the next pressButton-event is changed. When the operator presses next time a button, another corner point is taken to be dragged. The message transfer is in the same manner as in case of the fixed point. Now the operator can drag the cursor, which affects that the gui-object sends the drag-message to the parameterAdaption-object. New model parameters are calculated and the gui-object receives a redraw-message. When the operator releases the button, the interactive parameter adaption is done, and the gui-object changes to the entry state.

CORBA The **C**ommon **O**bject **R**quest **B**roker **A**rchitecture makes the operations of an object available for other objects on different systems. The object operations are available over system boundaries on different computers at different Internet locations, and they are implementable in different programming languages. A more detailed description about CORBA can be found in (Obj, 1996).

We made the interface of the Semi-Automatic Extraction System available by the CORBA mechanism. The interfaces can be accessed e. g. by a visible JavaBean™, which is responsible for visualization and receiving mouse events (cf. figure 11). A JavaBean™ is intended to be composed into applications by end users (Sun, 1997).

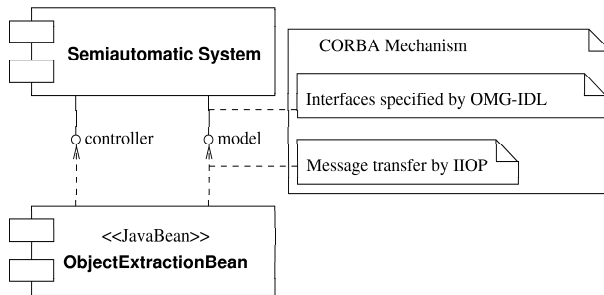


Figure 11: Component diagram of the semi-automatic extraction system implemented as a CORBA server, which is accessed by a JavaBean™. The JavaBean™ gets information for visualization from the model-interface and sends mouse events to the controller-interface. The signature of the interface-operations are specified by the OMG Interface Definition Language (OMG-IDL) and the messages are transferred by the Internet Inter-ORB Protocol (IIOP).

Java™ Object Serialization The Java™ object serialization mechanism is intended to serialize complete objects for persistence. The objects and all their aggregates are serialized including abstract references. We have applied this technique on a model with a reference to an abstract BRDF-object as described in section 2.3. Thus it is possible to store the abstract BRDF description in files and other applications are able to reconstruct the models and their reflectance behavior without knowing anything about the used BRDF model.

5 EXAMPLES

Figure 12 shows the complete Graphical User Interface of the Semi-Automatic Building Extraction System, whereas figure 13 shows a part the Semi-Automatic Building Extraction System loaded as JavaBean in the Java™-BeanBox, which is intended as an example for application development with JavaBeans. Figures 14-16 show a texture extraction example. The building model is attached with texture from two aerial images (cf. figure 14) and one close range image (cf. figure 15).

6 DISCUSSION

The Semi-Automatic System is intended to produce objects, which can be used in Geographic Information Systems. Objects represent a more complex kind of information than simple data. They are attached with operations, which are able to describe behavior and rules. Further, abstract concepts can be described by objects. Simple data consist only of numbers and strings in contrast to objects. It is difficult to model complex information in simple data structures.

We have depicted different ways of making objects accessible for other systems. The Semi-Automatic System is equipped with a

C++-API, a CORBA-interface and we have a rudimentary implementation of serializable Java™ objects. We are in a position to extract semi-automatically complex objects by keeping the object-oriented modeling. In this context the standardization of GIS-objects is of special interest. The Open GIS Consortium, Inc has recently released a Simple Features Specification for CORBA, which includes basically 2D geometry interfaces (Ope, 1998). This signifies, that we have to keep attention to future standardization developments with regard to 3D objects.

In section 3 we have specified several extraction methods, which are implemented within the Semi-Automatic Building Extraction System. In (Müller, 1997) we have presented a design pattern for the object-oriented modeling of matching tools. Some of the above mentioned extraction-tools are designed according to this pattern. Others are implemented, but not modeled according to a unique pattern, because they consist of legacy code. Especially the interactive methods are insufficient considered in the Matching Tool pattern. We need a model, which is flexible regarding to the type of extracted objects and to extraction method. With this modeling of extraction methods we are able to design a component based extraction system, which is so flexible that new components with new extraction methods can be easily integrated by the user.

It is desirable to integrate the Semi-Automatic System within another Applications, e. g. a GIS application. Since the Semi-Automatic System has interactive components we have to coordinate this with the respective application. The graphical user interface and the user-event handling should be designed in the same manner. A solution of this problem is given by the JavaBeans™ concept. We are able to create JavaBeans in the manner, we have shown in section 4. The Semi-Automatic System can be composed with other JavaBeans to complex applications. Precondition is, that other Systems support JavaBeans™ too. Because there are other software component models available ¹, we have to observe, which component model wins recognition in the future.

We have modeled reflectance in a reflectance-model independent manner using the facilities of object-oriented modeling. Applications, which handle reflectance, e. g. ray-tracers, need not to support a special reflectance model for the use of our objects. The extraction of reflectance is one of our future tasks.

For an optimal co-operation between a GIS and a Semi-Automatic Extraction System we suggest, that a GIS should support access by an object-oriented interface. Since extracted objects contain higher level information than simple data, this information would not get lost by fulfilling this request. New kinds of objects with individual implementations of their operations should be able to be integrated in a GIS. Our tests with Java™Object Serialization show, that this approach is principally feasible.

ACKNOWLEDGMENTS

The inspiration and support by Prof. Dr.-Ing. Wolfgang Förstner and Dr.-Ing. Eberhard Gülich are gratefully acknowledged. This research is supported by BMBF/DARA GmbH under Grant 50 TT 9733.

¹ e. g. ActiveX or DCOM for Windows based platforms

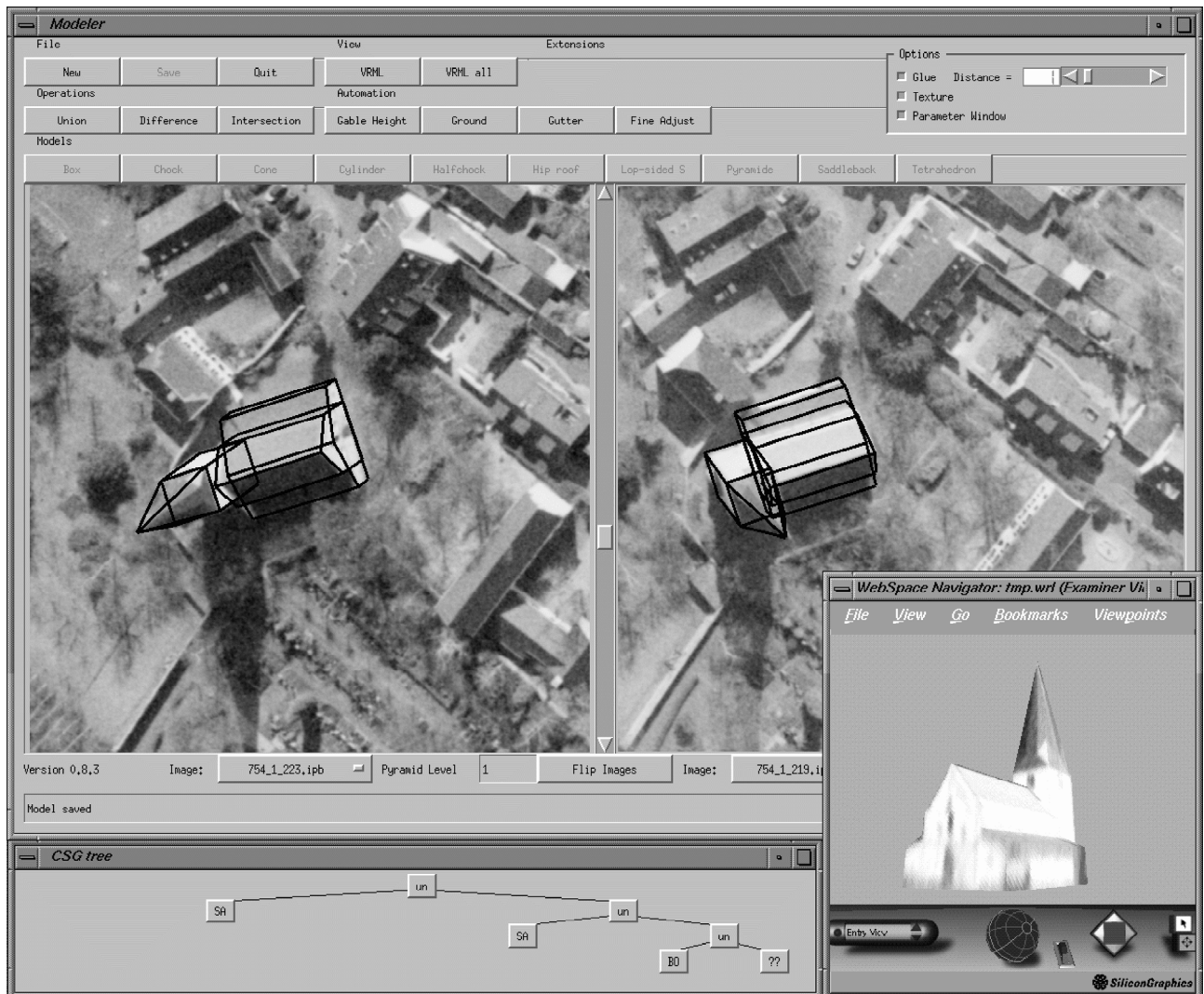


Figure 12: The Graphical User Interface of the Semi-Automatic System. The building models are projected as wire-frame models in the working area of the system and visualized with mapped texture in a VRML browser. The window at the bottom shows the structure of the CSG tree.

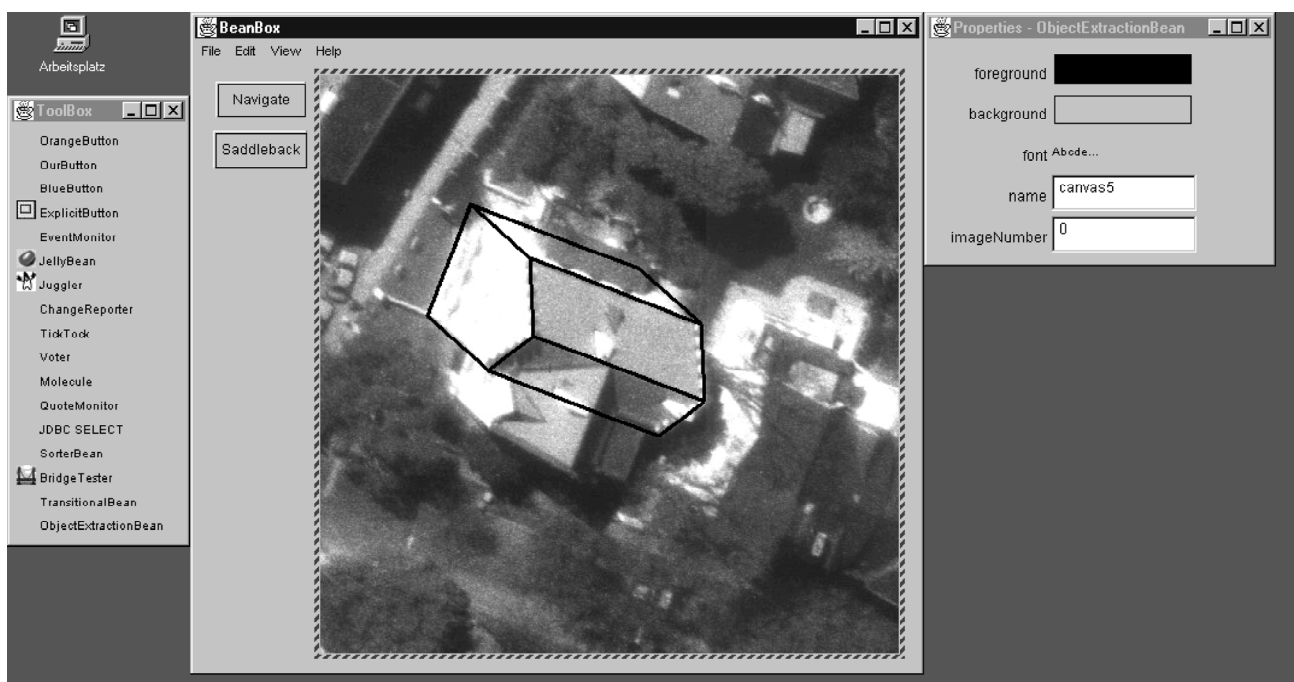


Figure 13: The Java™ BeanBox, an example application for JavaBeans™. The Semi-Automatic System is loaded in the working area as ObjectExtractionBean. This small application consisting of two buttons and the image area was composed in about 3-4 minutes to a runnable version. The remaining GUI elements belong to the BeanBox.

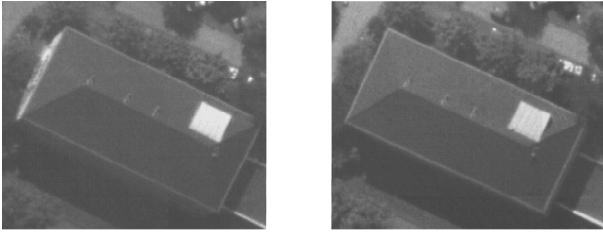


Figure 14: Aerial images



Figure 15: Close range image

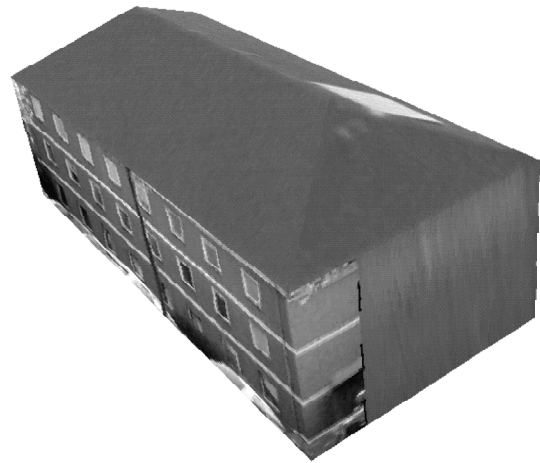


Figure 16: Texture extraction example. The texture is automatically extracted and mixed from two aerial images (figure 14) and one close range image (figure 15). Please note, that the wall to the right is only partly covered by the close range image and thus only partly mapped with texture from that image.

REFERENCES

- Booch, G., 1994. Object-oriented Analysis and Design. With Applications. Benjamin/Cummings Publishing Company, Inc.
- Englert, R. and Gülch, E., 1996. One-eye stereo system for the acquisition of complex 3D building descriptions. GIS.
- Fischler, M. A. and Bolles, R. C., 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. CACM 24(6), pp. 381–395.
- Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1995. Design Patterns. Addison-Wesley.
- Gülch, E. and Müller, H., 1997. Object-oriented software design in semiautomatic building extraction. In: Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision III, SPIE Proceedings, Vol. 3072.
- Hoffmann, C., 1989. Geometric and Solid Modeling. Morgan Kaufmann, Palo Alto, CA, USA.
- Koenderink, J. and vanDoorn, A., 1996. Bidirectional reflection distribution function expressed in terms of surface scattering modes. In: ECCV96, Springer-Verlag, pp. II:28–39.
- Läbe, T. and Ellenbeck, K.-H., 1996. 3D-wireframe models as ground control points for the automatic exterior orientation. In: Proceedings ISPRS Congress, Comm. II, Vienna, IAP Vol. XXXI.
- Läbe, T. and Gülch, E., 1998. Robust techniques for estimating parameters of 3D building primitives. In: Proceedings ISPRS Symposium, Comm. II, Cambridge.
- Lafortune, E. P. and Willems, Y. D., 1994. Using the modified phong reflectance model for physically based rendering. Technical Report CW 197, Department of Computing Science, K.U. Leuven. URL: <http://www.cs.kuleuven.ac.be/~eric1/Phong.html>.
- Lang, F. and Förstner, W., 1996. 3D-city modeling with a digital one-eye-stereo system. In: ISPRS Congress, Comm. IV, Vienna.
- Lang, F. and Schickler, W., 1993. Semiautomatische 3D-Gebäudeerfassung aus digitalen Bildern. Zeitschrift für Photogrammetrie und Fernerkundung 5, pp. 193–200.
- Müller, H., 1997. Designing an object-oriented matching tool. In: 3D Reconstruction and Modelling of Topographic Objects, International Archives of Photogrammetry and Remote Sensing, Vol. 32, ISPRS Commission III/IV, pp. 120–127.
- Müller, H., 1998. Experiences with semiautomatic building extraction. In: Third Course in Digital Photogrammetry, Institut für Photogrammetrie, Universität Bonn and Landesvermessungsamt Nordrhein-Westfalen, chapter 12.
- Obj, 1996. The Common Object Request Broker: Architecture and Specification. URL: <http://www.omg.org/corba/corbaio.htm>.
- Ope, 1998. OpenGIS Simple Features Specification For CORBA. Revision 1.0 edn.
- Rat, 1997. Unified Modeling Language. Version 1.1 edn. URL: <http://www.rational.com/uml>.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., 1991. Object-Oriented Modeling and Design. Prentice-Hall, Inc.
- Sun, 1997. JavaBeansTM. Version 1.01 edn. URL: <http://java.sun.com/beans>.
- VRM, 1997. The Virtual Reality Modeling Language. ISO/IEC 14772-1:1997, URL: <http://www.vrml.org/Specifications/VRML97>.