Context Sensitive Stylesheets for Scene Graphs

Erick Mendez [1] and Dieter Schmalstieg [2]

1 Graz University of Technology, Inffeldgasse 16a, Graz, Austria, mendez@icg.tugraz.at
2 Graz University of Technology, Inffeldgasse 16a, Graz, Austria, schmalstieg@icg.tugraz.at

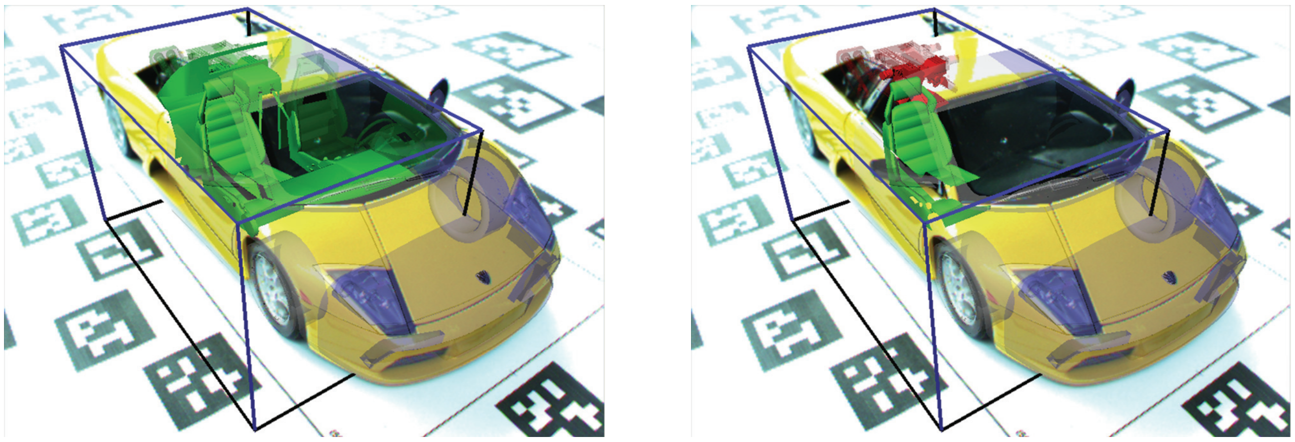


Fig. 1. Pieces of a scaled model car are dynamically mapped to the appropriate rendering styles inside a magic lens given contextual markup. On the left image an assortment of objects are mapped to the same rendering style, green. The image on the right shows a different mapping of rendering styles and scene objects. Such permutation of rendering styles and scene objects was not explicitly given by the application designer. The mappings are not based on a predefined scenegraph arrangement but on contextual markup.

Abstract – **Augmented Reality (AR) enables users to visualize synthetic information overlaid on a real video stream. Such visualization is achieved by tools that vary depending on the underlying data or the task at hand. While some tools distort, filter or enhance the explored information little or no work has focused on the separation of style definitions and their mapping to scene objects. We target this separation based on a context rich scenegraph. Our research allows the definition of visualization tools independent on the data to be visualized or the application that will use them. We show how contextual information may add another hierarchical dimension to scene objects, and how this may in turn be used by hierarchical style definitions –not unlike those of Cascading Stylesheets. We compare differences with previous work, we show a complex application example and we discuss the contribution of our work to AR.**

Index terms – **Augmented Reality, Information filtering, Layout Techniques**

I. INTRODUCTION

A frequent goal of Augmented Reality (AR) is to display abstract, occluded or otherwise imperceptible information linked to the real world. High density of such information leads to display clutter, so information filtering techniques have been investigated as a remedy. In practice many AR applications use an "object of interest" method where certain objects are of higher interest than others and, therefore, receive special treatment such as visual *style* changes - for example, cable damages or liver tumors may be highlighted. Such importance or interest is usually dynamically changing. For example, a tired tourist's interest may change from cultural landmarks to public transportation and the route back to the hotel.

This dynamic changing of the interest on objects increases the complexity of the application implementation and design. Traditionally, application design demands that designers have to decide on the range of possible objects of interest and their respective styles beforehand. If a scenegraph based implementation is used, the application code must be tightly coupled with the scenegraph data structure so that procedural calls can modify the rendering styles of the right portions of a scene graph. Fig. 1 shows an example of two different rendering style mappings applied to the same scene. Such mappings are not based on a knowledge of the scenegraph arrangement nor were they defined a priori by the application designer.

Ideally, application behavior leading to the definition of the styles, the selection of styles and the objects to which the styles are applied should be separate concerns, which can be designed and maintained independently. The application code should only control the mapping of styles to objects fitting a particular characterization, without having to touch the objects directly. In this way, a data driven application becomes feasible which allows content, styles and behavior to be varied independently in order to address a wide range of possible applications using the same AR system. This separation relies on two simple techniques:

1. *Context markup* is a technique of attaching free-formed context attributes to nodes in the scene graph hierarchy. The context is propagated downwards, and can be aggregated or modified as more context definitions are encountered during traversal. The context can either be user defined [14] or derived automatically, for example based on uncertainty estimation [11].
2. *Style maps* define a mapping of *context markup* to *style templates*. This mapping can be defined directly in the scene graph or attached from outside to the root of the scene graph. The latter approach allows defining and manipulating the *style template* independently of the scene graph. *Style maps* are organized hierarchically, similar to cascading style sheets for XHTML.

The binding of styles to objects happens very late, just before the traversal of the objects. Using context to connect styles to objects allows a clean separation of concerns and an arbitrary mix of local and global control of style. In particular, any procedural code that selects appropriate styles need not know details of the type, number or location of objects in the scene graph. The approach works particularly well when dynamic variation of styles is needed, such as in applications letting the user pick objects of interest interactively, for example, as in visualization techniques such as focus+context.

II. RELATED WORK

While most visualization work in general is concerned with visualization styles, in this work we are mostly interested in the strategies determining which styles to choose under what circumstances.

A lot of work lets a user choose a technique and then applies it uniformly to a particular dataset. For example, the original magic lens [1] [19] applies a style uniformly to everything contained in the lens. The context in this case is limited to the information whether a particular object is inside or outside the lens, but no further discrimination of objects is made. Although these kinds of tools are easy to develop they provide little help in the task of information filtering. For example, using a magnifying lens on a scene with a high amount of distracting information would result on the magnification of the overall scene without filtering excessive objects. To overcome this limitation [8] proposes to use interactive lens tools in AR in succession, but does not describe how the underlying management of objects works.

In our previous work [13] we introduce context sensitive magic lenses (CSML), which addresses a part of the problem discussed here by allowing the definition of styles depending on lens and object family. The work presented here extends CSML in the sense that a more general definition of styles and context is used, which can also be used to manage the assignment of styles independent of the existence of magic lenses to achieve other context-dependent visualization effects. Despite the fact that the technique described in this paper may be employed without a magic lens metaphor, we include this metaphor in order to emphasize the effects of rendering styles in the scene. Similar to our previous work we base our magic lens rendering on the technique described by [17] but based on a GPU programming language [10] for added performance.

Instead of contextual information, importance driven volume rendering [20] enriches a segmented volumetric dataset with a numeric value describing object importance which allows to infer visibility priority among the segmented objects and to emphasize important structures. Visualization is based on volume raycasting and considers the importance for every screen pixel separately, while our approach considers the context of objects as a whole.

Rule based visualization systems address the selection of style or content by dynamically applying a set of user-defined rules. A pioneering project in AR is [9] which generates technical illustrations for maintenance using an AR display. Another notorious work on information filtering for AR was introduced in [7], which uses spatial proximities to compute relative object importance given user properties such as tasks sets. These rule-based approaches are complementary to our work in the sense that they can easily be adapted to output style mappings or context attributes rather than directly modifying scene objects. Worth of mentioning is also the work by Götzelmannn et al. [6] who are able to modify the appearance of objects given connections present in database queries.

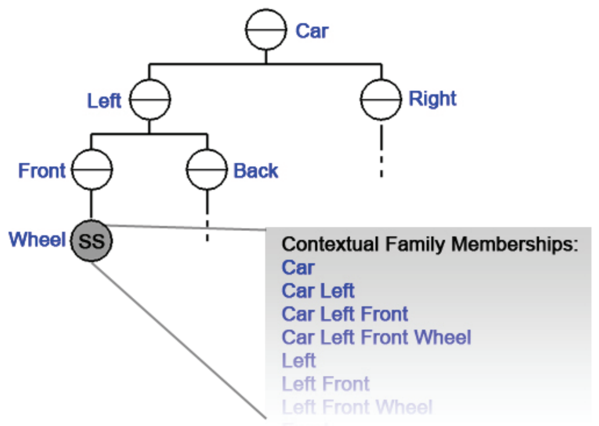
Decoupling of style and content has been tried before on the Graphical Kernel System by Bono et al [5]. That work, allowed the definition of graphical primitives independent of display devices and subject to the application programmer implementation. However, every graphical primitive still was bound to an attribute definition (called 'attribute bundle'), and could not be dynamically modified. Beach and Stone tackled this particular problem by defining graphical style sheets, as a way of describing design guidelines for an illustration [3]. That work allowed to change the appearance of primitives by simply specifying different styles which in turn were separated from content (called there 'format and content'). It allowed the edition of the style properties of a graphical node dynamically. However, the coupling of style and content had to be given explicitly. Our work allows said coupling to be given implicitly by context parameters, hence, allowing a higher level of control of the graphical style.

III. CONTEXT AND MAPPING DEFINE STYLE

All of the previously discussed techniques require a prior knowledge of the scene objects to be visualized, or have to perform expensive searches for objects. This limits the usability for large scenes and complex sets of categories. The technique presented here does not have these shortcomings. It is based on defining context for scene objects and *style maps* for visualization styles, and deriving the actual style for an object on the fly as a function of both context and mapping. Both context and maps are hierarchically organized and can be controlled independently.

III.1. Context markup

A context family of objects is defined as all those objects that have the same contextual markup. However, these objects do not have to belong to the same part of the scenegraph hierarchy – they can be scattered throughout the scene-graph and still be jointly affected according to the context family they belong to.



Contextual Family Memberships:
Car
Car Left
Car Left Front
Car Left Front Wheel
Left
Left Front
Left Front Wheel

Fig. 2. Illustration of the context family memberships of a *styled subgraph*. The memberships are all possible combinations of individual context attributes present during traversal. Therefore, the *styled subgraph* may be referenced by any of the combinatorial number of context families is member of.

An important consideration is that, unlike [13], the membership of an object in a particular context family is not explicitly given, but is the result of an aggregation of context attributes encountered while traversing the path from the scene graph's root node to the object itself. This mechanism works exactly like other traversal states such as transformations, where these are aggregated depending on their position in the graph. *Context attribute nodes* defining arbitrary key/value pairs are inserted as markups at arbitrary positions of the scene graph. Any *context attribute* encountered during the traversal is added to a set of current context attributes maintained during the traversal. Attributes can be overridden during the traversal if a second context node specifying the same attribute key is present.

The context family of a particular subgraph – marked up as a *styled subgraph* – is defined by the set of available contextual attributes at the moment of the subgraph's traversal. It is important to notice that every *styled subgraph* is member of several context families at the same time. This is because every context family may be composed of any combination of the partial or total context attributes present during traversal. We illustrate this in Fig. 2 where we show some of the context families the s*tyled subgraph* (marked as "SS") is member of. The membership to the different context families is not known but until the actual traversal since context values may be dynamically changing.

Further more, a simple change in the combination of context values used to reference a family may target hierarchically disconnected regions of the same scene graph. Fig. 3 illustrates this situation with a conceptual representation of a scene graph. At first, we reference objects belonging to the family "Car Left Front Wheel". Notice that this is a very restrictive combination of context values. Consequently, it yields only one object, signaled in red. However, if we were to remove one of the restrictions, namely "Left", this would reference objects in hierarchically separated parts of the graph (in blue). Traditionally, this mode of referencing would demand a detailed knowledge of the scenegraph or would require expensive search actions. However, this is easily achieved with our context markup since every object can retrieve its family memberships from the traversal state.

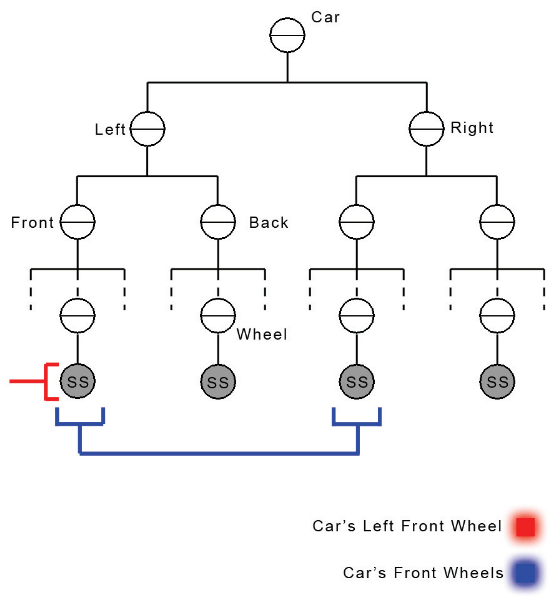


Fig. 3. An illustration on how a slight difference on the context values used for referencing may address objects in hierarchically separated portions of the scenegraph. If we reference only the front left wheel of a car (red) we get an isolated node. However if we remove the "left" restriction the objects whose context values match are in separated regions of the graph (blue).

III.2. Style templates and maps

A *style template* is a system resource given in the form of a named styling subgraph. This is typically composed of nodes controlling the visual appearance such as material definitions, textures, GPU shaders or even transformations. By allowing arbitrary scenegraphs as *style templates*, we target the widest possible control of visualization styles, including custom nodes and side effects. *Style templates* are used to influence the appearance of *styled subgraphs*. During the rendering traversal every *styled subgraph* is preceded by the first *style template* mapped to one of its context families.

The mapping of a context family to a *style template* is determined by another system resource, the *style map*. Every entry in the *style map* assigns a particular context family – given by a set of context attribute key/value pairs – to a *style template*. The *style map* is composed of individual nodes, each defining one mapping arranged hierarchically as part of the scene graph.
It is recommended that the content portion of the scene graph – containing the *styled subgraphs* marked up with context attributes – is kept in separate regions of the graph from the

style templates and *style maps*.

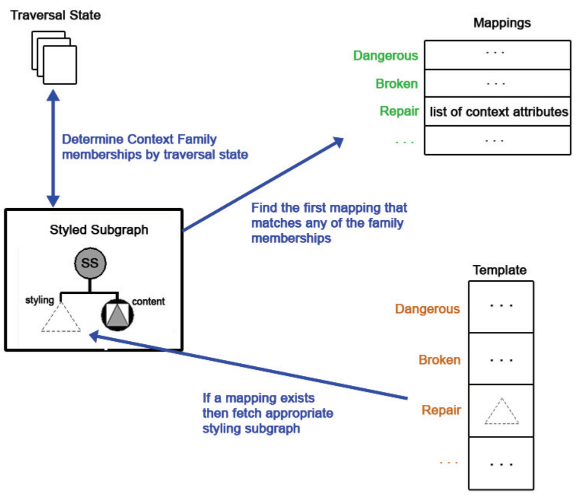

Traversal State
Mappings
Dangerous
Broken
Repair
list of context attributes
Determine Context Family memberships by traversal state
Find the first mapping that matches any of the family memberships
Styled Subgraph
SS
styling
content
Template
Dangerous
Broken
Repair
If a mapping exists then fetch appropriate styling subgraph

Fig. 4. During the traversal of the *styled subgraphs*, they check for the first mapping that matches their current context markup. If so, the appropriate styling subgraph is preceded to the content of the *styled subgraph*. Nested *styled subgraphs* are allowed and therefore incremental styling is possible.

Nonetheless the nodes of the *style map* are arranged hierarchically and their definition is extracted by traversal. This means that hierarchical styling rules can be constructed by defining more specific styles first, and then progressing to more general styles until a final default style with empty context. During the traversal of the scenegraph, every *styled subgraph* will check the defined mappings and then, depending on whether its own context markup is mapped to a template, it will fetch the appropriate styling subgraph defined by the template. This is illustrated in Fig. 4. Notice that the binding of rendering styles happens during traversal, therefore, we neither need to specify a style for every object, nor need to know the data structure arrangement of the scene. It must also be mentioned that the search for matching mappings does not depend on the combinatorial number of context family memberships of the *styled subgraph* but on the linear number of available mappings.

The typical overall scene graph structure consists of three subgraphs traversed in this particular order:

1. Subgraph containing all *style templates*
2. Subgraph containing all *style maps*
3. Set of subgraphs containing context attribute nodes and styled objects, called *styled subgraphs*

However, it is also permissible to interweave these three aspects at the expense of increased scene management complexity. The separation into distinct subgraphs mainly serves clarity and separation of concerns. In particular, all three aspects can be developed independently and by different designers as long as the style names and context attributes are consistent.

Fig. 5 shows a conceptual diagram of how the *style templates* and the context markup are linked by the style mappings. The hierarchical nature of the context markup allows the mappings to reference any context family available in the scene. The mappings in turn function like cascading stylesheets since every *styled subgraph* will progressively search for the first mapping that matches one of its context families. It is important to remark that the contents of a *styled subgraph* do not necessarily have to be atomic 3D objects, but can in turn be complex sub graphs or even nested *styled subgraphs*.

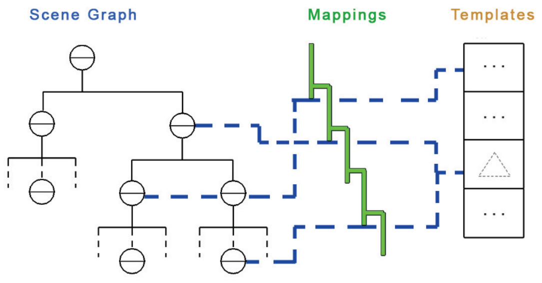


Fig. 5. This diagram shows how, conceptually, templates and context markup are linked together by the mappings. Notice that the hierarchical nature of the mappings allows them to function like cascading styles sheets. However, the separation of style and map allows them to reuse templates given different mappings.

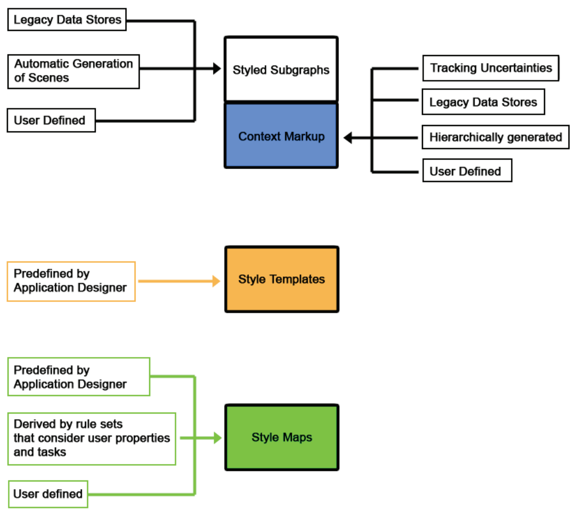


Fig. 6. Example of possible sources of information for all four building blocks. These vary from user defined to high complexity rule sets.

All four building blocks, *style templates*, *style maps*, *styled subgraphs* and *context markup* may be defined by different sources, ranging from user selection to automatic generation given higher rule sets. Fig. 6 shows a diagram of possible sources. For example, *styled subgraphs* may come from legacy data bases such as those from the Geographical Information Systems (GIS) community. GIS data is traditionally enriched with contextual information which may also serve as input for the *context markup*. Alternatively, tracking uncertainties like those described by Machado et al. [11] may also be used as context attributes. The style templates, however, are usually defined by application designers since visual styles and naming conventions may carry a semantic meaning, i.e. for example, the generation of a style for “Danger” is not easily generated automatically. Style maps in turn may come from a range of possible sources such as high complexity rule sets that consider user properties and tasks [7].

IV. ADAPTIVE AUGMENTED REALITY SCENARIOS

IV.1.A simple example

Consider an example application where the user has to repair a particular piece of a car. The pieces of this car have been enriched with contextual information such as whether they are interior or exterior, left or right, seats or engine and so on. This application allows users to mark a family of objects for repairing based on their contextual attributes. For example, the user may be interested in repairing all the wheels, or just the wheels on the right side of the car. At the same time dangerous objects are highlighted so that the user may be aware of potentially problematic situations.

This application defines templates for dangerous objects in red and repair objects in green. Notice that the designer of the application has not specified which objects are, for example, dangerous. This is because such semantic interpretation of dangerous objects may not be known until the moment of execution of the application. The following code achieves such templates:

```
StyleTemplates
{
 names  ["Dangerous", "Repair"]
 styles [USE RED, USE GREEN]
}
```

The mapping of styles and specific context families happens during the execution of the application. These mappings require a combination of context attributes, which define a context family, and a style name to which this context family will be mapped to. The following snippets achieve two different mappings:

```
StyleMap {
 templateName "Dangerous"
 keys ["Temperature", "Pressure"]
 values ["Hot", "High"]
}

StyleMap {
 templateName "Repair"
 keys ["Type"]
 values ["Wheel"]
}
```

Notice that the mappings of the "Repair" style to "Wheel" objects in this example is given explicitly. However, as mentioned before this mapping may be also achieved by rule engines that consider user tasks, for example.

The two last members of our framework are the *StyledSubgraph* and the *ContextAttribute*. The purpose of the *ContextAttribute* is to define a set of key value pairs at the moment of its traversal. The *StyledSubgraph* in turn defines the actual content to be traversed. The content itself may be a set of nested *StyledSubgraphs* with their own contextual attributes. The following snippet achieves a subgraph for a wheel that was last changed in January:

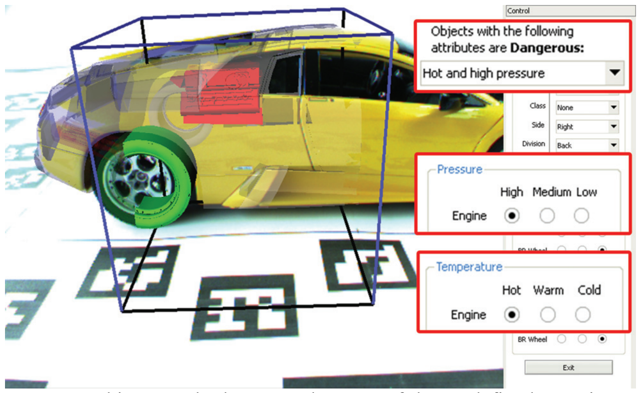
```
Separator {
  ContextAttribute {key "Type" value "Wheel"}
  ContextAttribute {key "Changed" value "Jan"}
  StyledSubgraph {
    content
    {
      # actual content goes here
      # may be a nested StyledSubgraph
    }
  }
}
```

Similar to the style mappings, the values of the contextual attributes in this example are also explicitly given but this may also be achieved by higher rule sets that consider external information or by specific combination of contextual values.

IV.2.Application Scenario

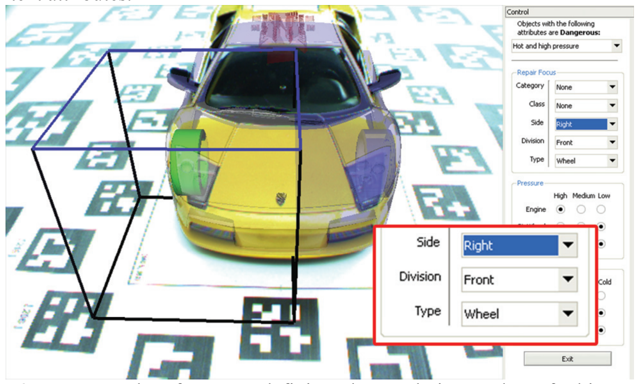
Based on the example outlined before, we describe now a more complex application scenario. This scenario includes three rendering style templates: dangerous, repair and neighbors. These three available styles change the appearance of the objects in the scene given mappings and context markup that may be given by the user implicitly, explicitly or defined a priori.

The templates in this case only modify the color and transparency attributes of those objects that fall inside a magic lens. Dangerous objects are colored in red when inside a lens and in red-half-transparent when outside. This allows the user to always be aware of dangerous objects regardless of whether they are in his work area or not. Objects for repairing in turn are colored as green when inside the lens and green-half-transparent when outside.



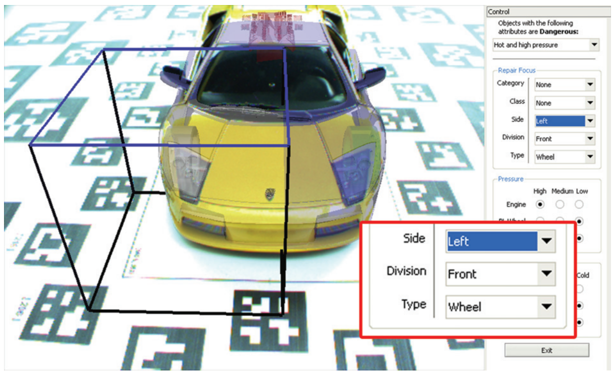
Hot and high pressure
Class None
Side Right
Division Back
Engine: [✓ High] Medium Low
Engine: [✓ Hot] Warm Cold

Fig. 7. In this example the user select one of the predefined mappings and effectively changes the families of objects in the scene by modifying their context attributes.



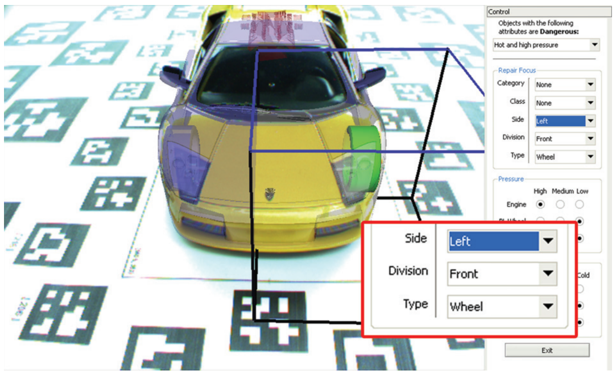
Hot and high pressure
Category None
Class None
Side Right
Division Front
Type Wheel
Engine: [✓ High] Medium Low

Fig. 8. Example of a user defining the rendering styles of objects by creating a customized mapping. In this example the right front wheel of the car is of the most interest to him. The magic lens is the black cubic wireframe.



Side Left
Division Front
Type Wheel

Fig. 9. The user has modified the mapping to the left wheel of the car. Dynamically all objects in the scene fetch the appropriate rendering styles.



Side Left
Division Front
Type Wheel

Fig. 10. A panning of the magic lens reveals that the appropriate rendering style is used.

Additionally, extra spatial information, such as the car's body, is shown in white-half-transparent. This provides depth cues for interior pieces.

The mapping of contextual markup and templates is done via different mechanisms. For example, for dangerous objects a number of predefined mappings are available. In Fig. 7 we show how the user has chosen one of the mappings with a combo box (top right). These mappings depend on temperature and pressure values of the *styled subgraphs*. These are not predefined during the application implementation but are user selected (middle and bottom right). In this case the car engine has this particular set of attributes and therefore is rendered accordingly.

In the case of repair objects the mappings are not predefined but the user is allowed to dynamically generate them given a selection of combo boxes. Figures 8 to 10 we show a sequence of steps that illustrate the dynamic bindings of styles. Fig. 8 shows that the user is interested in repairing the front right wheel of the car. Fig. 9 then shows how the mapping has been changed to the left wheel. Objects in the scene whose context markup is being referenced change their rendering style. In this case the mapping has been changed from the right to the left wheel. At this point in time the left wheel is shown green-half-transparent since it does not fall inside the lens. Once the lens has been panned Fig. 10 and the wheel falls inside it then the appropriate rendering styles are used. Notice that throughout the whole sequence, an outline of the car body is shown in half transparent. This outline is, however, different from that of Fig. 7 which involves a different part of the car (the rear). This is because the last template, called neighbors, is being mapped implicitly by the user given his position in relation to the car. By doing this we filter distracting artifacts that may needlessly increase the depth complexity of the scene.

In this application the three style templates have been in use providing different visual information to the user. Some provide visual depth cueing, others highlight problematic pieces and others draw the attention to focus objects. This means that three tasks are running concurrently: spatial context, danger awareness and priority targets. These three tasks are highly reminiscent to those of [7].

IV.3. Expected Error as Context

An interesting value to use as contextual markup is that of the expected error of fiducial tracking systems, such as ARToolkitPlus[*]. A number of tests have been carried out to detect the accuracy of fiducial tracking in position and orientation [1] [12]. A simple error function was presented in [1] which depends on the distance and angle to the marker. For the sake of illustration we will use this function at a constant distance of 30 cm. For this distance the tracking expected error is:

- High error between 90 and 85 degrees
- Low error between 85 ad 20
- High error between 20 and 0 degrees

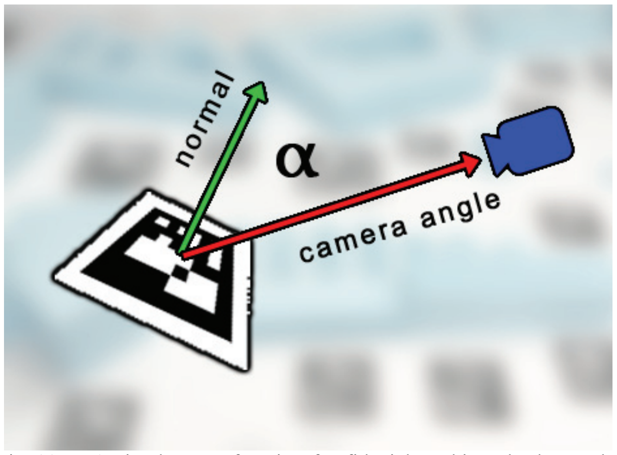


Fig. 11. A simple error function for fiducial tracking checks on the angle between the camera and the normal of the marker

This angle is the orientation vector of the marker relative to the camera as returned by the fiducial tracking and the normal of the marker, this is illustrated in Fig. 11

Separation onto two distinct families is a trivial task in this case. Objects whose position is being tracked by a marker will also set a contextual value called "Error" to the value returned by the previous function. In other words, objects will belong to one of two families: "Error=High" or "Error=Low" depending on the angle they have with the marker's normal and the values listed above.

[*] http://studierstube.org/handheld_ar/artoolkitplus.php

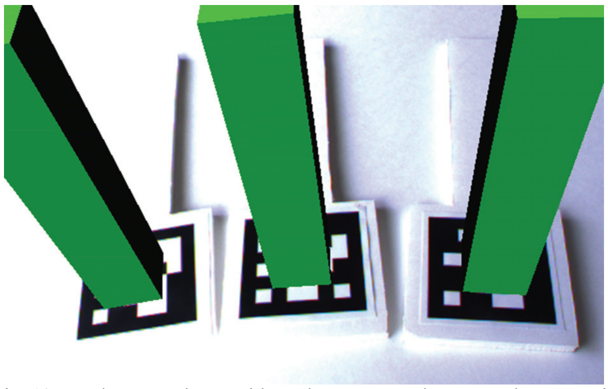


Fig. 12. Three markers with a low expected error. The error is calculated from the angle between the vector from the camera to the marker and the marker's normal.

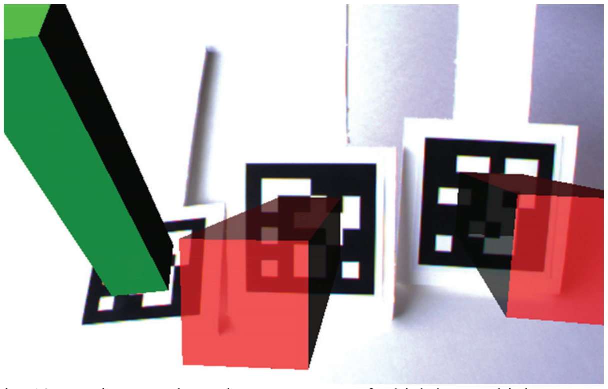


Fig. 13. Three markers shown, two out of which have a high expected error. This expected error is used as a context attribute, effectively creating context markup families on the fly given tracking information.

We create two templates called "Reliable" and "Unreliable". The first is set to a green color and the second to red half transparent. The mappings are quite straight forward: Objects with a high error should be mapped to the "Unreliable" template and objects with a low error to the "Reliable" template.

Fig. 12 shows three objects tracked by three different fiducial markers. Because of the angle between the camera and the normal of the markers, all of the objects have a low expected error. Fig. 13, however, shows that two objects have a high expected error. This is due to a change in their orientation that reduced the angle between the normal and the camera.

Notice that the grouping does give any guarantee about the quality of the tracking. This grouping is only based on a measure of the expected error. Although this is a simple example, contextual markup may be used by more robust uncertainty and error models.

A potential application of such context markup is to apply, for example, a scaling factor to the object. The resulting rendering of the object after such scaling may be used to identify "screen real estate" regions of the object similar to those used by Bell et al. [2] for annotation. By having a real state region defined not only as a function of the object's shape, but of a mixture of the object's shape, its tracking uncertainty and its contextual markup may lead to more powerful annotation techniques. Fig. 14 shows an example application of this idea. Objects in this example create "screen real estate" regions that are used by annotations for correct placement. On the upper image objects are not scaled because they are accurately tracked. On the lower image we simulate a situation with bad tracking (exemplified by the orthogonal angle to the markers). The rendered object in the scene is mapped to a down scaling rendering style that guarantees a real state region useful for labeling (in blue).

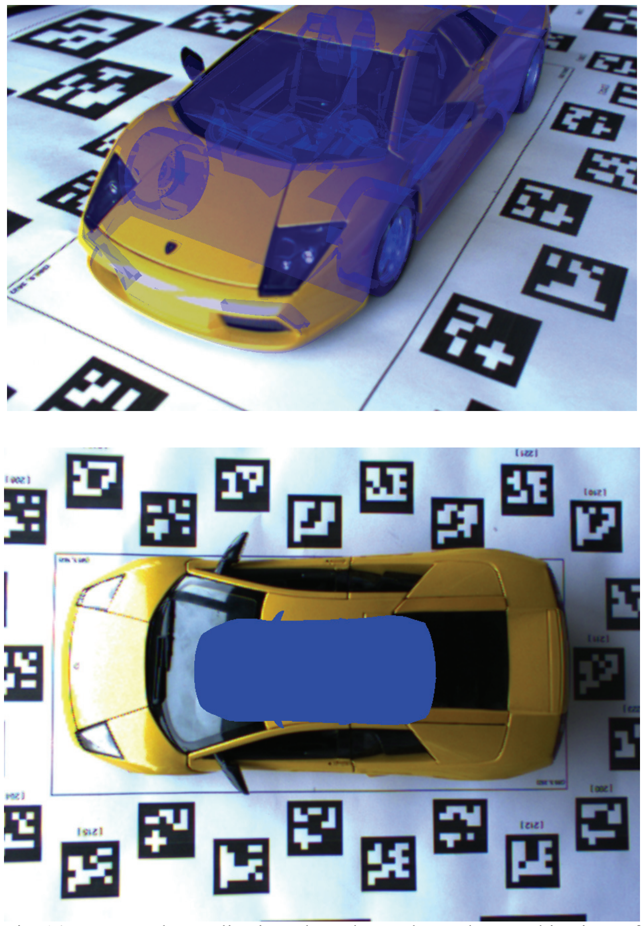


Fig. 14. Example application that shows how the combination of contextual markup derived from tracking uncertainties and rendering styles may aid annotation. Above: the object has a high quality tracking. Below: exemplified bad tracking. Objects with context markup defined by the accuracy of the tracking apply a scaling rendering style that reduces their "screen real estate". This real estate (in blue) may be used by annotations for correct placement.

V. DISCUSSION

Now that all the building blocks have been presented, it is important to underline the difference between traditional, context sensitive and markup/mapping magic lenses. To exemplify these differences we use an AR setup based on data extracted from a GIS database. This data includes building outlines, electricity and gas lines. Additionally we have placed one extra landmark for aided complexity. A magic lens has been placed in the scene and to avoid image clutter we do not display objects that fall outside the lens.

The original work of Bier et al. allowed styles to be assigned to objects depending whether they were inside or outside the magic lens. An example of this type of lens renders all objects inside it, for example, in red. This type of lens will render in red objects in the scene with which it interacts whether these objects are of interest to the user or not. It will also render these objects regardless of their data structure arrangement or tagged contextual information. Although these kinds of tools are easy to develop they provide little help in the task of information filtering. For example, it would be impossible to visually isolate objects of interest without a prior knowledge of the scenegraph data structure.

In our previous work [13] we showed how objects in a scene enriched with contextual information could be affected differently given the same magic lens. That work showed how objects could be grouped by context in addition to hierarchically. That work, however, required a prior definition of the available styles and did not separate these from the mappings with contextual families. Further more, membership to a contextual family was given explicitly by a single context attribute without hierarchical characteristics.

The technique proposed in the current paper highly extends the original CSML in two major parts: It allows a clear separation of style definitions and style mappings, and it allows contextual attributes to be hierarchically inherited. The two approaches mentioned before would not be capable of dynamically binding rendering styles to objects given properties such as user interest or tasks.

Context-aware computing is popular in the ubiquitous computing (Ubicomp) community for creating adaptive user interfaces, but not so much in the AR community. Through the simple and well-known mechanism of using markup to add context to scene graphs, we hope to encourage researchers and developers relying on existing scene graph solutions to adopt our approach. Like Ubicomp, AR relies on sensor networks which can also be a source of context information.

Markup can be retrofitted to existing scene graphs. While our implementation is based on Coin3D[†], there is nothing in our work that precludes the quick integration of context markup in other scene graph toolkits. It is largely compatible not only with standard scene graphs, but also with the modeling and interaction techniques typically used in conjunction with scene graph based engines.

The fact that application designers let users influence the styles of objects implicitly instead of explicitly will reduce the amount of interaction required. This may translate into simpler user interfaces, such as hands-free interfaces, which play a prominent role in AR.

The separation of *styled subgraphs* from *style templates* with explicit mapping allows developing scene graphs independently of style design and visualization policies. *Style maps* share the capability of XHTML cascading style sheets that a graphics designer can determine visualization styles once. These styles can then be applied to a variety of application and arbitrary content. This makes it easier to design complex content – a requirement not unique to AR but very relevant if one wants to incorporate legacy data sources such as from the geographic information systems (GIS) we are currently working with.

The examples shown in this paper are actually a part of a bigger project called *Vidente*[‡], in which AR visualization of subsurface features is investigated. It is throughout this project that the idea of using templates and context based interaction was born. The results of the presented research are already being integrated in the framework of *Vidente*.

ACKNOWLEDGEMENTS

Our thanks go to Grintec GesmbH for providing data for the GIS examples. This research was funded in part by FWF (Y193), FFG (BRIDGE 811000) and the European Union (European Union MRTN-CT-2004-512400).

REFERENCES

[1] Abawi, D.F. Bienwald, J. Dorner, R. , "Accuracy in optical tracking with fiducial markers: an accuracy function for ARToolKit," In: Mixed and Augmented Reality, 2004. ISMAR 2004. pp: 260- 261

[2] Bell, B., Feiner, S., and Höllerer, T. "View management for virtual and augmented reality," Proc. UIST '01 (ACM Symp. on User Interface Software and Technology), Orlando, FL, November 11-14, 2001 (CHI Letters, vol. 3, no. 2), 101-110.

[3] Beach Richard and Stone Maureen, "Graphical style towards high quality illustrations", In SIGGRAPH Comput. Graph. 1993. pp. 127-135

[4] Bier Eric, Stone Maureen, Pier Ken, Buxton William, DeRose Tony, "Toolglass and Magic Lenses: the see-through interface," In proceedings SIGGRAPH 1993, pp. 73-80

[5] Bono, P.R.; Encarnacao, J.L.; Hopgood, F.R.A.; ten Hagen, P.J.W., "GKS---The First Graphics Standard", IEEE Computer Graphics and Applications 1982, pp. 9-23

[6] Götzelmannn Timo, Hartmann Knut, Nürnberger Andreas, and Strothotte Thomas; 3D Spatial Data Mining on Document Sets for the Discovery of Failure Causes in Complex Technical Devices; 2nd International Conference on Computer Graphics Theory and Applications (GRAPP'07), 2007.

[7] Julier S., Lanzagorta M., Baillot Y., Rosenblum L., Feiner S., and Hollerer T. Information filtering for mobile augmented reality. In Proc. ISAR 2000, pages 3--11, Munich, Germany, October 5--6 2000

[8] Looser J., Billinghurst M., Cockburn A., "Through the looking glass: the use of lenses as an interface tool for Augmented Reality interfaces," In Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and SouthEast Asia (Graphite 2004). 15-18th June, Singapore, 2004, ACM Press, New York, New York, pp. 204-211

[9] Feiner Steven, Seligmann Dorée, "Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations," The Visual Computer 8(5&6) , 1992, pp. 292-302

[10] Mark W., Glanville R. S., Akeley K., Kilgard M.. Cg: A system for programming graphics hardware in a C-like language. Proceedings SIGGRAPH 1993.

[11] Machado Coelho Enylton, MacIntyre Blair, and Julier Simon (2004) "OSGAR: A Scene Graph with Uncertain Transformations" In Proceedings International Symposium on Mixed and Augmented Reality (ISMAR04), November 2-5, 2004, Washington, D.C., USA.

[12] Malbezin, P., Piekarski, W., Thomas, B., "Measuring ARToolkit accuracy in long distance tracking experiments," In: 1st Int'l AR Toolkit Workshop. (2002)

[13] Mendez E., Kalkofen D., Schmalstieg D., "Interactive Context-Driven Visualisation Tools for Augmented Reality," In

[†] www.coin3d.org

[‡] www.vidente.at

Proceedings of ISMAR 2006, October 22-25, 2006, Santa Barbara, California, USA. pp. 209-218

[14] Reitmayr Gerhard, Schmalstieg Dieter, "Flexible Parameterization of Scene Graphs," In proceedings IEEE Virtual Reality Conference 2005 (VR'05), 2005, pp. 51-58

[15] Reitmayr Gerhard, Schmalstieg Dieter, "Semantic World Models for Ubiquitous Augmented Reality," Proceedings of Workshop towards Semantic Virtual Environments' (SVE 2005), 16th-18th March 2005

[16] Reitmayr Gerhard and Schmalstieg Dieter, "Collaborative Augmented Reality for Outdoor Navigation and Information Browsing," Proc. Symposium Location Based Services and TeleCartography, Vienna, Austria, January, 2004, Available as technical report TR-188-2-2003-28, TUWien

[17] Ropinski Timo and Hinrichs Klaus, "Real-Time Rendering of 3D Magic Lenses having arbitrary convex Shapes," In proceedings International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, 2004, pp. 379-386

[18] Schmalstieg Dieter and Reitmayr Gerhard, "The World as a User Interface: Augmented Reality for Ubiquitous Computing," In Location Based Services and TeleCartography (eds. Georg Gartner, William Cartwright, Michael P. Peterson), pp. 369-382, Springer Berlin Heidelberg New York, ISBN 3-540-36727-6, 2007.

[19] Viega John, Conway Matthew, Williams George, Pausch Randy, "3D Magic Lenses," In proceedings ACM Symposium on User Interface Software and Technology, 1996, pp. 51-58

[20] Viola Ivan, Kanitsar Armin, Gröller Meister Eduard, "Importance-Driven Volume Rendering", in proceedings of IEEE Visualization 2004, pages 139-145

[21] Wang L., Zhao Y., Mueller K., Kaufman A., "The Magic Volume Lens: An Interactive Focus+Context Technique for Volume Rendering,", In IEEE Visualization, 2005

Erick Mendez is a research engineer and PhD student at the Graz University of Technology. His research interests include context information and visualization techniques for augmented reality systems. Mendez received his MSc in computer science from The George Washington University and his BSc in computer Science from the Benemerita Universidad Autonoma de Puebla. Contact him at mendez@icg.tugraz.at..

Dieter Schmalstieg is a professor of virtual reality and computer graphics at the Graz University of Technology, where he directs the Studierstube research project. His research interests include augmented reality, virtual reality, distributed graphics, 3D user interfaces, and ubiquitous computing. Schmalstieg received a Doctorate of Technology in computer science from the Vienna University of Technology. He's an editorial advisory board member on Computers & Graphics, a member of IEEE ISMAR's steering committee, chair of the Eurographics Working Group on Virtual Environments, and advisor of the K-Plus Competence Center for Virtual Reality and Visualization in Vienna. Contact him at schmalstieg@icg.tugraz.at.