5

Kaleidoscape: A 3D Environment for Querying ODMG Compliant Databases

Norman Murray, Carole Goble and Norman Paton
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL, UK
{murrayn, carole, norm}@cs.man.ac.uk

Abstract

Kaleidoscape is a three dimensional (3D) implementation of a data-flow oriented visual query language, which has been implemented in 3D to examine the advantages and disadvantages of such an interface paradigm over current WIMP GUIs. This paper describes a version of Kaleidoscape that allows the user to construct queries from within a 3D environment. These queries are then translated into the ODMG standard textual query language OQL for evaluation, the results of which can be viewed and browsed from within the Kaleidoscape environment.

Keywords

visual query language, 3D, OQL, ODMG, results visualisation

1 INTRODUCTION

Kaleidoquery (Murray, Paton & Goble 1998) is a visual query language for ODMG (Object Data Management Group) compliant object databases (Cattell et al. 1997). Queries created with Kaleidoquery can be translated into ODMG version 2.0 Object Query Language (OQL). This allows implementations of Kaleidoquery to be used with any object database that conforms to the ODMG standard.

Kaleidoscape is the 3D implementation of Kaleidoquery that was designed using the framework in (Murray, Goble & Paton 1998). We chose to implement Kaleidoquery in a 3D environment in order to assess any benefits gained and problems found when using 3D visualisations that utilise the human perception system's ability to understand full three dimensional volumetric space, and to examine the impact of differing forms of hardware ranging from the standard monitors and desktop mice to head mounted displays, 3D mice and

Visual Database Systems 4 Y. Ioannidis & W. Klas (Eds.)

auto-stereoscopic displays. A belief exists that conventional direct manipulation WIMP interfaces, while working well for some tasks, may be a limiting factor for others (Raskin 1997, van Dam 1997). Interface hardware is now reasonably common and becoming established, allowing 3D location information, gesture and speech recognition.

In this paper, we begin with a brief introduction to the Kaleidoquery language and the Kaleidoscape interface architecture. We then examine the implementation of the interface by looking at the presentation of the database schema, navigation within the environment, the construction of a query, and the display of the results. Finally, we discuss some conclusions drawn from experience with the interface, and list future work required on the interface to support the user in constructing queries in a 3D environment.

2 KALEIDOQUERY DESIGN PHILOSOPHY

The Kaleidoquery language is fully described in (Murray, Paton & Goble 1998). In this section we give a brief introduction to and explanation of the design of the language.

Visual query languages attempt to bridge the gap of usability for users over standard textual query languages and environments. Many problems with textual query languages have been identified, and these include: a steep learning curve; common semantic and syntactic errors; the structure of the database classes, attributes and relationships is not readily available to users; increasing complexity when specifying the order of boolean operators with parentheses as the query grows (Borgman 1986, Greene et al. 1990, Michard 1982); and the differences in meaning between the English and boolean logic meaning behind the *and* and *or* operations (Greene et al. 1990, Michard 1982).

In creating Kaleidoquery we attempted to solve the above problems. By creating a visual query language and generating a display of the database schema we hope to solve the first three problems. To solve the problem with the boolean operators we utilised a filter flow approach to query specification, substantially extending earlier uses of the approach (Shneiderman 1991).

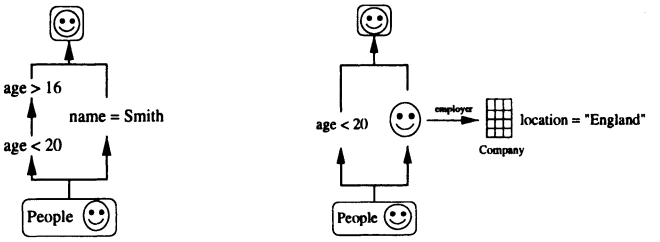
Providing the advantages of a visual query language over a textual language, Kaleidoquery attempts to solve these problems using the above mentioned methods and provides:

1. an unusually powerful visual query language for OODBs, supporting the full functionality of OQL (Cattell et al. 1997),
2. compliance with the ODMG model version 2.0 and consistency with OQL with its well understood language constructs plus direct support for evaluation,
3. a filter flow oriented visual model,
4. a separation of the tasks of writing the query constraints and organising the structure and ordering of the results.

In Figure 1 (a) we see the Kaleidoquery representation of the following OQL query:

```
select p
from p in People
where (p.age < 20 and p.age > 16) or (p.name = "Smith")
```

The instance information flows up through the constraints from the set of instances (extent) called *People* of the class *Person*, visualised with a simple person icon, ☺. With *and*, the constraints on the age follow one after the other. As the information flows through the constraint on each age, instances that do not satisfy the constraint are *filtered* out. The boolean *or* is visualised by the branching of the filter flow paths. The results of the constraints on each half of the *or* query (the two constraints on age *or*-ed with the constraint on the name) are combined on completion of the *or*. The results of the query flow into a new group of *persons*, visualised at the top of the query.



(a) Visualising *and* and *or* (b) Visualising navigation in a constraint

Figure 1 Example visual queries

Figure 1 (b) visualises the OQL query:

```
select p
from p in People
where p.age < 20 or p.employer.location = "England"
```

On the left of Figure 1 (b) we can see how a constraint has been formed through navigation from one class to another related class. To place a constraint on a person's company, the user has to navigate from the *Person* class to the *Company* class. The navigation along the relationship is visualised as a horizontal arrow with the relationship name located above the arrow. After the arrow the icon of the related class is displayed, ▦. From the *Company* class we can select the attribute location and place the constraint that it be equal to "England".

3 KALEIDOSCAPE IMPLEMENTATION

Kaleidoscape has been implemented on a Silicon Graphics workstation using the OpenInventor library for display of the 3D graphics, and the Minimal Reality (MR) toolkit to provide access to head mounted displays (HMD) and 3D mice. The Kaleidoscape interface is capable of exploiting HMDs, stereo projectors, auto-stereoscopic displays and monitors for visual display, and 3D mice, desktop mice and the keyboard for user input. The database interface doesn't talk directly to the database but talks to another module that communicates directly with the database, in this case O_2. This has the advantage that when a different object database is to be used it is only the program that talks to the database that needs to be changed.

Queries in Kaleidoscape are stored as graphs. Each class on which constraints have been placed has a set of nodes associated with it, with each node containing a description of the constraint. Together, each of these nodes forms a graph defining the structure of the query.

To create the OQL equivalent of the visual query we traverse the constraint node graph. The *select* clause of the OQL query comes from the attributes that the user has selected for inclusion in the results. The *from* clause is defined by the extent or query results that flow into the data-flow. To construct the *where* clause of the OQL the constraint node graph of the class is traversed, with the appropriate OQL being generated. If the constraint node contains a pointer to a related class constraint node graph then this is translated into OQL. This continues until the final node in the graph is translated. The OQL generated can then be sent to an ODMG compliant database for evaluation.

4 KALEIDOSCAPE ENVIRONMENT

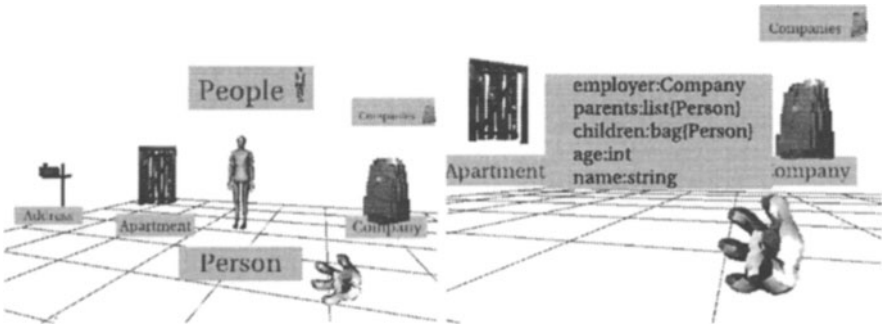
4.1 Schema Display and Navigation

The schema is composed of particular classes, chosen from the database that the interface is communicating with and the extents that hold the instances of the database classes. We need to visualise these classes within the environment so that the participant can browse the schema and compose queries, see Figure 2 (a) *. In visualising the schema we will examine the following areas: class and extent presentation; the landscape environment; and navigation.

(a) Class and Extent Presentation

The class visualisation is composed of the name of the class and an appropriate 3D icon. For example, Figure 2 (a) depicts the classes, *Address*, *Apartment*,

*This plus other figures of the interface are screen dumps from a session using Kaleidoscape on a Silicon Graphics workstation.



(a) Database schema in Kaleidoscape

(b) Attribute display of the class *Person*

Figure 2 Database schema and attribute display

Person and *Company*. The name of the class is always oriented towards the participant so that as they move around the schema, the class name is always visible. We have chosen to display both the class name and a visualisation of the class, as studies have shown that icons with a textual description give better comprehension than purely textual or pictorial icons (Kacmar & Carey 1991). As the participants become familiar with the class visualisations they will associate the icon with the class and in this way will not have to read the class name. Also, if the class is being viewed from a distance the class name may not be entirely legible but the participant may be able to recognise the class icon.

Located above each of the classes are any extents associated with the class. These contain the instances of the class that are stored in the database. This is visualised as the extent name along with the class icon showing what type of instance the extent contains. For example, in Figure 2 (a) we can see that the classes *Person* and *Company* have the extents *People* and *Companies* located above their respective class visualisations. Along with the external iconic view of the class and its name, the class also has to display its attributes. Examining the attributes of a class can be done by selecting the class with the 3D mouse or the desktop mouse. In Figure 2 (a) we see a visualisation of a *virtual hand* that moves in response to the movements of the 3D mouse. This can be used to select objects in the environment, or the participant can select items using the desktop mouse cursor.

When a participant selects a class, the selection mimics an animated hyperlink, in that the user is moved towards the class in animated steps, first by rotating the user to directly face the class and then *flying* them towards the class visualisation which alters to show the internal attribute structure of the class as they approach it.

At present the attribute visualisation consists of a textual display, as shown in Figure 2 (b). This display consists of a list containing each attribute name along with its associated type. For example, in Figure 2 (b) we see that the *Person* class is composed of five attributes, including an *employer* that is of the type *Company*, and a list of *Persons* containing the *parents* of a *Person*. At present, relationships between classes are only visualised by the type of the attribute being displayed textually alongside the attribute name. For example, in Figure 2 (b) we see that the attribute *employer* is of type *Company*. A more direct visual cue to the related class could easily be added, for example, as arrows linking the related classes.

(b) Landscape Environment

As can be seen in Figure 2 (a) the classes are laid out on a plane or landscape, which is depicted as a simple tiled floor. The entire database interface is depicted within the same environment – that is the database schema, the queries that the user is constructing and the results of these queries are all located in the same 3D environment. With conventional interfaces the database, query and results are usually located in separate windows, but by using our landscape metaphor, results of queries can be located *alongside*, *behind* or even *inside* the query, alleviating the task of window management. Also, queries on similar topics could be clustered together in the environment.

The Kaleidoquery language lends itself to constructing queries from other queries. As the queries are constructed, the results of the query are visualised at the top of the query where they can be used as input to another query.

(c) Navigation

We have seen how the database schema is visualised and how the participant can navigate to classes by selection. Movement around the database environment can be performed using the keyboard or mouse to move forwards, backwards, rotate, etc. The participant can also use the 3D mouse to move around the environment. When in use, the user can see the 3D mouse represented as a hand in the environment, as seen in Figure 2. The mouse has 5 buttons, 3 placed along the top of the mouse and 2 trigger buttons placed on the front. As the 3D mouse is moved, or the user alters the orientation of the mouse, the *virtual hand* moves to reflect the real movements. To move through the environment, the participant orientates their hand to point in the direction that they wish to move (with the virtual hand this direction is depicted as the direction the fingers are pointing). As they move they can continue to alter the orientation of their hand to change the direction in which they are moving. The user can also browse the schema by moving between related classes. The class *Person* in Figure 2 (b) contains the attribute *employer* that is of type *Company*. If we wish to view the attribute details of the *employer* class, then we select the attribute *employer* and the user is au-

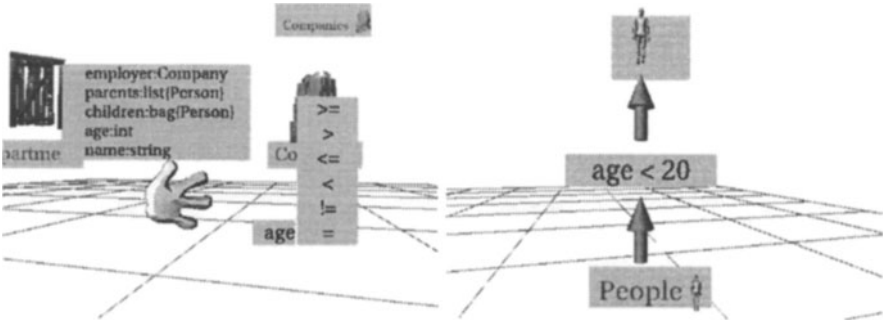
tomatically navigated towards this related class in the 3D environment. This can be described as an animated hyperlink.

4.2 Query Creation and Visualisation

In the previous sections we saw how the schema is displayed and how the participant can navigate through the schema and browse the classes, attributes and relationships. To compose a query involves: selecting the classes that we wish to query over; placing constraints to restrict the instances of interest; and selecting what attributes to see in the results. We shall examine how these tasks are performed by looking at the construction of some example queries.

(a) Simple Query Constructs

In this section we show how to build a simple query to find all the instances of the type *Person* in the extent *People* with the constraint that they are aged less than 20. To begin construction of this query we select the class on which we are interested in placing the constraint, in this case the *Person* class. As previously described, this results in the display of the attributes of the *Person* class, as seen in Figure 2 (b). To construct a constraint on this class, we select the attribute on which we wish to place the constraint, in this case the *age* attribute. On selection of this attribute we are presented with the operations that can be applied to that attribute, as shown in Figure 3 (a).



(a) Some operations available on the *age* attribute

(b) Query with a single constraint on age

Figure 3 Class attributes and a simple query

From this list, the operation < can be chosen, and the scalar value *20* entered via the keyboard. This results in the Kaleidoquery shown in Figure 3 (b). We can also see in Figure 3 (b) that the interface has chosen to use the *People* extent as input for the filter flow query as it is the only extent on the

class *Person*. From the visual query, we see that the instances of the extent *People* flow up to the constraint and the instances that satisfy the constraint are allowed to flow into the results visualised at the top of the query. To view the results of a query, or of any extent in the schema, the user simply selects the results or extent visualisation, whereupon they are transported to another environment displaying the instances (the results environment is described in Section 4.3).

(b) *and* and *or*

If we are to place more constraints on the query in Figure 3 (b) then we will need to combine them with the equivalent of boolean operations *and* and *or* in OQL. We showed how this is visualised using the filter flow method in Section 2.

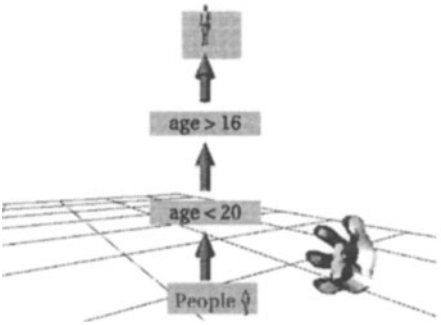
Figures 4 (a), (b) and (c) show how the boolean operators are visualised. In Figure 4 (a) the two constraints on the attribute age are aligned on the same flow. This means that the instances of the extent *People* first pass through the constraint *age* $<$ *20*. The instances that satisfy this constraint flow upwards to be filtered by the next constraint *age* $>$ *16*. The instances that also satisfy this constraint are allowed to flow into the results visualisation at the top of the query. Where constraints follow one another on the same data-flow, this is equivalent to the boolean operation *and*.

In Figure 4 (b), we can see that the data-flow has divided into two. The instances from the *People* extent follow both paths. The result of this query is a union of the instances that have *age* $<$ *20* and those with *name* $=$ *Smith*. Finally, in Figure 4 (c) we see a query containing a combination of constraints that have been *and*-ed and *or*-ed. This is the same query as was defined in Section 2 and shown in Figure 1 (a).

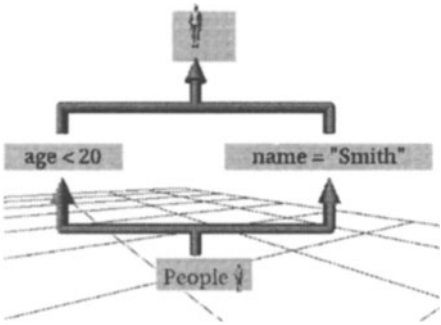
To construct the above query the user has to indicate how they want to combine their constraints. The first constraint, shown in Figure 3 (b), is placed in the query automatically as there is only one choice for its placement. As additional constraints are constructed, the user needs to select where in their query they want to place the constraints. To do this the user selects the data-flow pipe where they want the query to appear using their *virtual hand* or the desktop mouse, using different mouse buttons in the present implementation to distinguish between *and* and *or*.

(c) Path Expressions

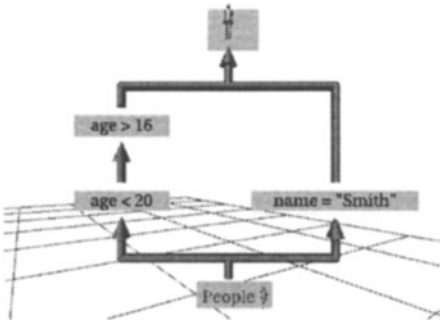
Previously, we have only seen queries being constructed on the *scalar* attributes of the class that was first selected, such as the types *string* and *integer*. We can also apply queries using the relationship attributes of a class. For example, we see that the *Person* class shown in Figure 3 (a), along with the *scalar* attributes *age* and *name*, has the related attribute *employer* that is a link to an instance of the class *Company* and also contains links to *parents* and *children*.



(a) *and*-ed query



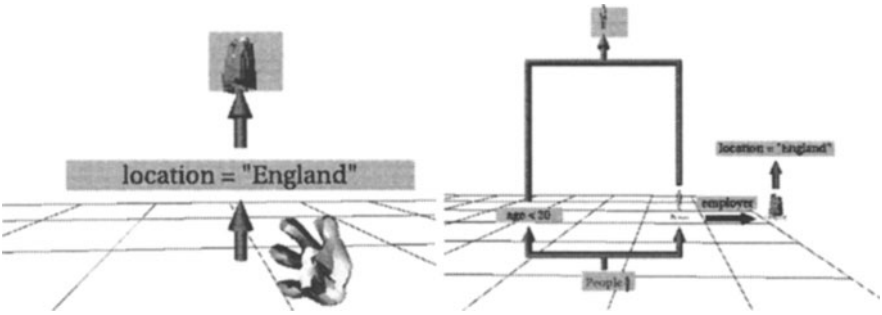
(b) *or*-ed query



(c) *and* and *or*

Figure 4 Queries showing *and* and *or*

In Section 4.1 (c) on Navigation we saw how movement between related classes was performed, where the user selected the related class in the list of the class's attributes, and was animated towards the related class in the database schema. When the user selects a ***relationship*** attribute, they can either perform some aggregate operation on the ***related*** attribute, e.g. count, sum, etc., or move to the related class and place constraints on the attributes of the related class. When the constraints on the related class have been completed, we can then combine these related constraints with our original query by selecting where in the data-flow we wish to place the related constraints. For example, if we wish to construct the query to find all *persons* that are aged less than 20 or have employers located in "England", we first build the *age < 20* constraint as seen in Figure 3 (b). We then select the *employer* attribute of the *Person*'s class, seen in Figure 3 (a). This moves us to the *Company* class and we are shown its attributes. From this we can build the constraint *location = "England"* shown in Figure 5 (a). From this figure it can be seen that at



(a) Query built on *Company*

(b) Completed query with constraint on *simple* and *complex* attributes

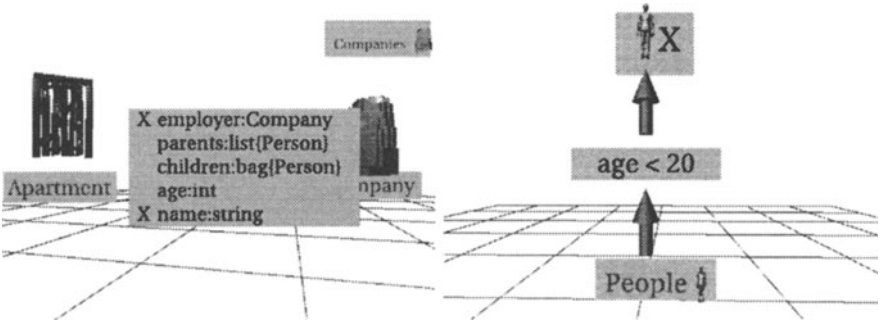
Figure 5 Building constraints on a relationship attributes

present there is no data-flow input to the query. This is because we have not yet placed the constraint within the query that we are constructing on the *Person* class. Satisfied with the construction of this constraint we can return to the *Person* class and select the data-flow where we wish to place our new constraint. The data-flow input to the constraint we created on the class *Company* shown in Figure 5 (a) is the instances of the *employer* attribute of the *Persons* in the extent *People*.

(d) Results Selection

With the query as shown in Figure 5 (b), the results can be seen to have flowed along the data-flow arrows of the query to the icon displayed at the top of the query. As we selected no attributes for inclusion in the result, the results of the query will be formed of the complete instances. If only certain key attributes of the class are required in the results, the user can select these by placing a cross beside the attributes for inclusion, as shown in Figure 6 (a). If we look at Figure 6 (b) we see how this is visualised in the query. The visualisation of the results at the top of the query has a cross annotated next to the icon representing the *Person* class to indicate that only a selection of the complete attributes of the class are in the results.

Other functionality of the query language OQL, implemented in Kaleidoscape is catered for in a similar manner to the previously mentioned functions, i.e. after the user selects an object in the environment they are presented with a list of the available operations on that object. For example, if they select a related class they can navigate towards it to browse it and construct some constraints as previously defined, or they could use the related class in a membership, universal or existential quantification test. If the user selects the results visualisation of a query, then the tasks that they can perform over



(a) Selected attributes shown in the class attribute view

(b) Query showing how attributes are shown selected

Figure 6 Selecting attributes for inclusion in the results

the results include viewing the results, as discussed in the following section, or structuring or ordering the results.

4.3 Results Visualisation

The results of a query can be viewed in the 3D environment, or can alternatively be presented textually in a terminal window. To gain access to the results environment users select the result or extent visualisation that they wish to view. On selection they are moved towards this visualisation. As they *hit* the instance visualisation they *enter* the instance visualisation where the results are displayed.

Currently the results are located in the 3D space according to the values of certain scalar attributes of the instances. Three of these values are used to position the instance in 3D space with a fourth being used for the colour of the instance. The user is allowed to alter the four attributes that have been chosen to visualise the instance data to alter the results display. Of course, the user also has the option of only using one or two of the dimensions for spatially locating the results. The user can also scale each of the individual axes to expand or compress the results.

In Figure 7 we see how the attributes *name*, *age* and *number of children* have been used to distribute the instances. Colour has been used to represent whether the parents of the instances are present in the database. Each of the instances are automatically displayed according to the type of the attribute, so strings are sorted alphabetically, collections by size, etc. The *number of children* axis has been scaled down as can be seen by comparison with the other axes.

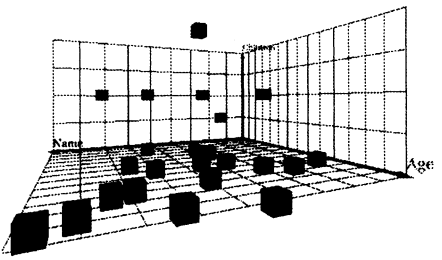


Figure 7 The current results environment

5 RELATED WORK

Visual query languages have attempted to bridge the gap of usability for users (for a survey see (Catarci et al. 1997)). Forms based query languages such as QBE (Zloof 1977), present the database structure as tables or forms into which queries can be placed. Graph based query languages (e.g. Guidance (Haw et al. 1994)) have the advantage over forms style interfaces in that they can directly represent relationships within the structure of the database and the query. Icon based languages (e.g. Iconic Browser (Tsuda et al. 1990)) represent database concepts pictorially and allow direct manipulation of icons to represent queries. Multi paradigm query interfaces also exist to allow the user to pick and choose or alternate between interface styles, (Doan et al. 1995).

Current graphical query languages to ODMG compliant object databases are limited to Quiver (Chavda & Wood 1997) and GOQL (Keramopoulos et al. 1997). Only two simple examples of the Quiver query language are given in (Chavda & Wood 1997), but preliminary evaluations show that it is easier to use than the standard textual OQL interface. GOQL is a graph style query language and a complete description of its constructs is given in the paper. Neither of the query languages describe the interface that has been implemented to support the visual language, and neither exploit 3D environments.

The three main areas of interest in 3D interfaces to databases are schema display, query creation and display and results visualisation. WINONA (Rapley & Kennedy 1994) displays the schema using either a hierarchal or circular wall visualisation, with querying being limited to simple form-based string searches. Lyberworld (Schweickert & Hemmje 1996) uses a cone tree structure for selecting predefined queries, and distributes the results inside a sphere according to their relevance to key attributes located on the surface of the sphere. Q-PIT (Benford & Mariani 1994) locates and visualises the instances of the database in a 3D environment according to the mapping of the in-

stances attributes. AMAZE (Boyle & Gray 1995) comes closest to a complete 3D interface to a database with the schema and results being visualised in 3D, although queries are entered via a forms interface. AMAZE's query language is also less powerful than Kaleidoscape lacking self-join and visual expression of aggregate operations.

6 CONCLUSIONS AND FURTHER WORK

With Kaleidoquery and Kaleidoscape we set out to create a more usable visual version of OQL and to create an interface for viewing the database schema, building a query and viewing the results in three dimensions. We also wished to examine the utility of currently available display hardware, such as auto-stereoscopic displays, 3D projectors, head mounted displays, and interaction hardware and techniques, such as 3D mice.

We have completed the task of creating a visual OQL, and this paper reports on the early implementation of the 3D environment. Presently most of the Kaleidoquery language has been implemented within the 3D environment and the schema can be browsed and queries can be built, translated into OQL and passed to a database, with the results being presented in a separate 3D environment.

We now need to concentrate on taking more advantage of the 3D environment, and on creating tools to help the participant in the environment. We shall see how we are going to tackle each of these problems in turn.

6.1 Exploitation of the 3D Environment

As mentioned previously, Kaleidoquery was not designed specifically for implementation in a 3D environment. We shall now look at how we can enhance not only the Kaleidoquery language but also the complete environment to take further advantage of the added dimension.

The Kaleidoscape environment is primarily composed of the schema, query and results visualisations. We shall examine how each of these visualisations could be enhanced.

Database Schema Currently the database schema visualises the classes laid out on a landscape in user-defined groups. Other aspects of the database schema that could be visualised are the relations between classes which could be shown with arrows linking the relevant classes. Class hierarchies could be depicted in a similar manner.

The Query We have to be careful when altering the query so that it takes advantage of the extra dimension. We do not want the query to become unreadable when the user directly views the query due to occlusion. Parts of the query could obscure other sections making it unreadable without either the

user moving in the environment or the user manipulating the query, which could annoy the user. Having said that, the extra dimension does have its uses. With complex queries with many filters and complex results structures, the data-flow lines could overlap, which can create confusion in a 2D environment as seen in some graph displays. By using the extra dimension we would hope to alleviate this problem.

We could also used the third dimension for displaying complex queries over many classes. The main query would be displayed along two axes, going from left to right and bottom to top. Filters on classes other than the main query class could be displayed on the horizontal plane, using depth to abstract out sections of the query.

We could also have layered queries. When building a query we may want to produce a new query that is only slightly different from a previous query. Rather than copying this query, the queries could share similar filters – depth would be used for displaying the different filters. In this way we know that the queries are almost identical. In addition to showing queries in a layered form we could show the results of layered queries on the same results axes to highlight the differences in the results.

Results Visualisation With visualising the results we have used the three dimensions for spatially locating the results. The visualisation could be enhanced by allowing the user to alter the range of the axes to filter the instances shown. For example, along the y axis could be plotted the number of children a person has. This could range from 0 to 6. If the user is only interested in people with children then they could alter this range to cover 1 to 6, thereby removing all instances of people with no children.

The results environment could also be extended to allow other attributes of the instances to be used to specify their values, e.g. size, shape, opacity, spin speed, etc. could be used for mapping the values of the instance attributes to their visualisation.

6.2 User Support

The ideas for this future work come from both our earlier VRML prototype evaluations and observations of the use of the current system. Some of these ideas include:

Map A map would aid navigation through large schemas, as well as giving an overview of the database schema.

Message Bar This would contain information on the current state of the task the user is carrying out.

Query History and Workspace This would allow for the storage of queries and their use in further queries, allowing for refinement of query results.

Query Generation Currently the participant begins building their queries from within the database schema rather than within the class instances. We could examine allowing the participant to begin creating their query from within the instance data environment as done in Spotfire (Ahlberg 1996), although Spotfire can at present only handle databases containing a single table.

Our current interface still requires entry of information via the keyboard, although movement and selection of artifacts in the environment can be accomplished via the keyboard, desktop mouse or 3D mouse. We could also look at alternate methods of information entry such as speech and hand held chord keyboards which have proved popular with wearable computers (Starner et al. 1997).

At present we are reviewing the interface informally through heuristic and expert evaluation. When we have completed construction of the Kaleidoscape environment we will perform a more rigorous user evaluation of the interface. To fully acknowledge the benefits and downfalls of the 3D environment we could compare it with a WIMP implementation of Kaleidoquery, although at present this does not exist. The completed Kaleidoscape interface and its evaluation will be reported in a future paper.

REFERENCES

Ahlberg, C. (1996), 'Spotfire: An Information Exploration Environment', *SIGMOD Record* **24**(4), 25–29.

Benford, S. & Mariani, J. (1994), Virtual environments for data sharing and visualisation - populated information terrains, *in* 'Proceedings of the 2nd International Workshop on User Interfaces to Databases'.

Borgman, C. L. (1986), 'The User's Mental Model of an Information Retrieval System; An Experiment on a Prototype Online Catalog', *International Journal of Man-Machine Studies* **24**, 47–64.

Boyle, J. & Gray, P. M. D. (1995), The Design of 3D Metaphors for Database Visualisation, *in* 'Proceedings of Visual Database Systems', Chapman and Hall, pp. 185–202. Stefano Spaccapietra, Ramesh Jain (Eds.).

Catarci, T., Costabile, M. F., Levialdi, S. & Batini, C. (1997), 'Visual Query Systems for Databases: A Survey', *Journal of Visual Languages and Computing* **8**, 215–260.

Cattell, R. G. G., Barry, D., Bartels, D., Berler, M., Eastman, J., Gamerman, S., Jordan, D., Springer, A., Strickland, H. & Wade, D. (1997), *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, Inc.

Chavda, M. & Wood, P. (1997), Towards an ODMG-Compliant Visual Object Query Language, *in* M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos & M. A. Jeusfeld, eds, 'Proceedings of 23rd

International Conference on Very Large Data Bases', pp. 456–465.

Doan, D. K., Paton, N. W. & Kilgour, A. C. (1995), 'Design and User Testing of a Multi-paradigm Interface to an Object-Oriented Database', *ACM SIGMOD Record* **24**(3), 12–17.

Greene, S., Devlin, S., Cannata, P. & Gomez, L. (1990), 'No IFs, ANDs, or ORs: A Study of Database Querying', *International Journal of Man-Machine Studies* **32**, 303–326.

Haw, D., Goble, C. & Rector, A. (1994), GUIDANCE: Making it Easy for the User to be an Expert, *in* 'Proc. 2nd Int. Workshop On Interfaces to Database Systems', Springer-Verlag, pp. 19–43. P. Sawyer (Ed).

Kacmar, C. J. & Carey, J. M. (1991), 'Assessing the Usability of Icons in User Interfaces', *Behaviour and Information Technology* **10**(6), 443–457.

Keramopoulos, E., Pouyioutas, P. & Sadler, C. (1997), GOQL, a Graphical Query Language for Object-Oriented Database Systems, *in* 'Basque International Workshop on Information Technology', pp. 35–45.

Michard, A. (1982), 'Graphical Presentation of Boolean Expressions in a Database Query Language: Design Notes and an Ergonomic Evaluation', *Behaviour and Information Technology* **1**(3), 279–288.

Murray, N., Goble, C. & Paton, N. (1998), 'A Framework for Describing Visual Interfaces to Databases', *To be published in the Journal of Visual Languages and Computing* .

Murray, N., Paton, N. & Goble, C. (1998), Kaleidoquery: A Visual Query Language for Object Databases, *in* 'Proceedings Advanced Visual Interfaces'.

Rapley, M. H. & Kennedy, J. B. (1994), Three Dimensional Interface for an Object Oriented Database, *in* 'Proc. 2nd Int. Workshop On Interfaces to Database Systems', Springer-Verlag, pp. 143–167. P. Sawyer (Ed).

Raskin, J. (1997), 'Looking for a Humane Interface: Will Computers Ever Become Easy to Use?', *Communications of the ACM* **40**(2), 98–101.

Schweickert, T. & Hemmje, M. (1996), A Graphical User Interface to the Object-Oriented Database System VODAK on the Basis of the Generic Visualisation Toolkit Lyberworld, *in* 'Proc. 3rd Int. Workshop On Interfaces to Database Systems', Springer-Verlag. J. B. Kennedy and P. J. Barclay (eds.).

Shneiderman, B. (1991), Visual user interfaces for information exploration, *in* 'Proceedings of the 54th Annual Meeting of the American Society for Information Science', Learned Information Inc., Medford. NJ, pp. 379–384.

Starner, T., Mann, S., Rhodes, B., Levine, J., Healey, J., Kirsch, D., Picard, R. W. & Pentland, A. (1997), 'Augmented Reality through Wearable Computing', *Presence* **6**(4), 386–398.

Tsuda, K., Hirakawa, M., Tanaka, M. & Ichikawa, T. (1990), 'Iconic Browser: An Iconic Retrieval System for Object-Oriented Databases', *Journal of Visual Languages and Computing* **1**(1), 59–76.

van Dam, A. (1997), 'Post-WIMP User Interfaces', *Communications of the ACM* **40**(2), 63–67.

Zloof, M. (1977), 'Query-By-Example: A Data Base Language', *IBM Systems Journal, Vol. 4* pp. 324–343.

7 BIOGRAPHY

Norman Murray received his B.Sc. in Computer Science from Aberdeen University in 1994 and M.Sc. in Human Computer Interaction from Queen Mary and Westfield College, the University of London in 1995. Since then he has been at University of Manchester conducting research towards a Ph.D. on the topic of "Interface Environments to Object Databases" funded by the EPSRC. His research interests include visual query languages, virtual reality, and information visualisation.

Carole Goble is a senior lecturer in information systems at the University of Manchester, where she co-leads the Information Management Group. She was a research associate at Manchester from 1982 and joined the faculty in 1985. Her research interests include hypermedia and multimedia information systems, description logics for managing data and metadata, user-interfaces to databases, and distributed information systems.

Norman Paton is a senior lecturer in information systems at the University of Manchester, where he co-leads the Information Management Group. Prior to this, he was a lecturer at Heriot-Watt University from 1989-1995, and a research associate at Aberdeen University from 1986-1989. His research interests include active databases, deductive object-oriented databases, user-interfaces to databases, and distributed information systems.