

# Evolutionary Query Processing, Fusion and Visualization

Shi-Kuo Chang<sup>1</sup>, Weiying Dai<sup>1</sup>, Stephen Hughes<sup>2</sup>, Prasad S. Lakkavaram<sup>1</sup> and Xin Li<sup>1</sup>

<sup>1</sup>Department of Computer Science, University of Pittsburgh

<sup>2</sup>Department of Information Science, University of Pittsburgh  
chang@cs.pitt.edu

## Abstract

An *evolutionary query* is a query that may change in time and/or space. For example when an emergency management worker moves around in a disaster area, a predefined query can be executed repeatedly to evaluate the surrounding area in order to find out objects of threat. Depending upon the position of the person or agent and the time of the day, the query can be modified either manually or automatically. A novel approach for sensor-based evolutionary query processing is described, taking into consideration query optimization because most sensors can generate large quantities of spatial information within short periods of time. The sensor dependency tree is used to facilitate query optimization. Through query refinement one or more sensor may provide feedback information to the other sensors. Combined with visualization, the approach facilitates multi-sensor information fusion in dynamic environments.

## 1. Introduction

There is an important new class of queries for emerging applications such as emergency management, pervasive computing and situated computing [Nakashima02], which requires novel query processing, fusion and visualization techniques. We will call this class of queries *evolutionary queries*. An evolutionary query is a query that may change in time and/or space. For example when an emergency management worker moves around in a disaster area, a predefined query can be executed repeatedly to evaluate the surrounding area in order to find out objects of threat. Depending upon the position of the person or agent and the time of the day, the query can be quite different. The person or agent who issues the query is called the *query originator*. Depending upon the temporal/spatial coordinates of the query originator, an evolutionary query may be modified either manually or automatically.

We describe a novel approach for sensor-based evolutionary query processing using the sensor dependency tree. In our approach, one or more sensor may provide feedback information to the other

sensors through query refinement. The status information such as position, time and certainty can be incorporated in multi-level views and formulated as constraints in the refined query. In order to accomplish sensor data independence, an ontological knowledge base is employed. The results of query processing are visualized so that the user can also modify and/or refine the query.

This paper is organized as follows. The sensor data dependency tree is introduced in Section 2. Section 3 describes the query refinement approach. Query processing example and query optimization technique are presented in sections 4 and 5, respectively. An experimental prototype is described in Section 6. Section 7 describes our overall approach for query processing, fusion and visualization.

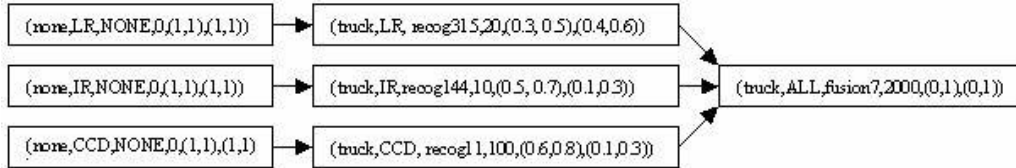
## 2. The Sensor Dependency Tree

In database theory, query optimization is usually formulated with respect to a query execution plan where the nodes represent the various database operations to be performed [Jarke84]. The query execution plan can then be transformed in various ways to optimize query processing with respect to certain cost functions. In sensor-based query processing, a concept similar to the query execution plan is introduced. It is called the *sensor dependency tree*, which is a tree in which each node  $P_i$  has the following parameters:

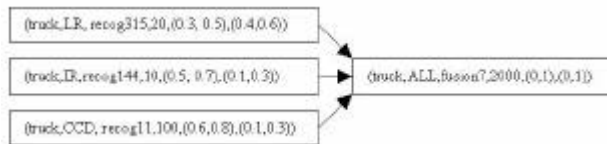
- obj\_type: the object type to be recognized
- source: information source or an operator such as fusion
- recog\_alg: object recognition/fusion algorithm to be applied
- time: estimated computation time for recognition/fusion
- recog\_cr: certainty range [min,max] for object recognition
- norecog\_cr: certainty range for object non-recognition
- sqo: spatial coordinates of the query originator
- tqo: temporal coordinates of the query originator
- soi: space-of-interest for object recognition/fusion
- toi: time-of-interest for object recognition/fusion

Query processing is accomplished by the repeated computation and updates of the sensor dependency tree. During each iteration, one or more nodes are selected for computation. The selected nodes must not be dependent on any other nodes. After the

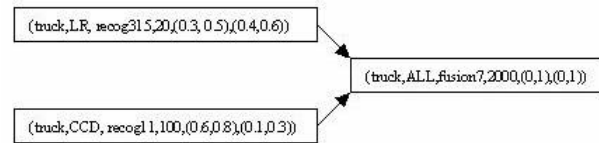
computation, one or more nodes are removed from the sensor dependency tree. The process then iterates. As an example, by analyzing the initial query, the following sensor dependency tree T1 is constructed:



Next, we select some of the nodes to compute. For instance, all the three leaf nodes can be selected, meaning information will be gathered from all three sources. After this computation, the processed nodes are dropped and the following updated sensor dependency tree T2 is obtained:



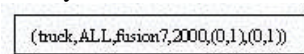
We can then select the next node(s) to compute. Since IR has the smallest estimated computation time, it is selected and recognition algorithm 144 is applied. The updated sensor dependency tree T3 is:



In the updated tree, the IR node has been removed. We can now select the CCD node and, after its removal, select the LR node. The tree T4 becomes:



Finally the fusion node is selected. The tree T5 is:



After the fusion operation, there are no unprocessed (i.e., unselected) nodes, and query processing terminates.

### 3. Query Refinement

In the previous section, a straightforward approach of sensor-based query processing is described. This straightforward approach misses the opportunity of utilizing incomplete and imprecise knowledge gained during query processing.

Let us re-examine the above scenario. After IR is selected and recognition algorithm 144 applied, suppose the result of recognition is not very good, and only some partially occluded large objects are

recognized. If we follow the original approach, the reduced sensor dependency tree becomes T3 of the previous section.

But this misses the opportunity of utilizing the incomplete and imprecise knowledge gained by recognition algorithm 144. If the query is to find unoccluded objects and the sensor reports only an occluded object, then the query processor is unable to continue unless we modify the query to find occluded objects. Therefore a better approach is to refine the original query, so that the updated sensor dependency tree T6 becomes:



This means recognition algorithm 144 is applied to detect objects in an space-of-interest soi-23. After this is done, the recognition algorithm 315 is applied to recognize objects of the type 'truck' in this specific space-of-interest. Finally, the fusion algorithm fusion7 is applied.

Given a user query in a high-level language, the natural language, a visual language or a form, the query refinement approach is outlined below, where *italic words* indicate operations for the second (and subsequent) iteration.

**Step 1.** Analyze the user query to generate/*update* the sensor dependency tree based upon the ontological knowledge base and the multi-level view database that contains up-to-date contextual information in the object view, local view and global view, respectively.

**Step 2.** If the sensor dependency tree is reduced to a single node, perform fusion operation (if multiple sensors have been used) and then terminate query processing. Otherwise build/*refine* the  $\delta$ -query based upon the user query, the sensor dependency tree and the multi-level view database.

**Step 3.** Execute the portion of the  $\delta$ -query that is executable according to the sensor dependency tree.

**Step 4.** Update the multi-level view database and go back to Step 1.

As mentioned above, if in the original query we are interested only in finding un-occluded objects, then

the query processor must report failure when only an occluded object is found. If, however, the query is refined to "find both un-occluded and occluded objects", then the query processor can still continue.

Evolutionary queries and query processing are affected by the spatial/temporal relations among the query originator, the sensors and the sensed objects. In query processing/refinement, the spatial/temporal relations must be taken into consideration in the construction/update of the sensor dependency tree. The temporal relations include "followed by", "preceded by", and so on. The spatial relations include the usual spatial relations, and special ones such as "occluded by", and so on [Lee92].

#### 4. Query Processing Examples

As a simple example of a  $\sigma$ -query [Chang98, Chang99], the *source* R consists of time slices of 2D frames. To extract three pre-determined time slices from the source R, the query in mathematical notation is:  $\sigma_t(t_1, t_2, t_3) R$ . The meaning of the  $\sigma$ -operator in the above query is "select", i.e. we want to select the time axis and three slices along this axis. The subscript t in  $\sigma_t$  indicates the selection of the time axis. In the SQL-like language the  $\Sigma$ QL query is expressed as:

```
SELECT t
CLUSTER t1, t2, t3
FROM R
```

A new keyword "CLUSTER" is introduced, so that the parameters for the  $\sigma$ -operator can be listed, such as  $t_1, t_2, t_3$ . The word "CLUSTER" indicates that objects belonging to the same cluster must share some common characteristics (such as having the same time parameter value). A cluster may have a sub-structure specified in another (recursive) query. Clustering is a natural concept when dealing with spatial/temporal objects. To facilitate query formulation, an alias (the name of a variable) can be given to a cluster.

In what follows we present a more complicated example to illustrate the processing of queries that may involve one or more sensors, and the possibility for query optimization. Suppose the query is:

**Query 1:** "Find two frames from a video frame sequence where a frame containing a white bird comes after a frame containing a red bird."

We can construct the query, which extracts all the object pairs OBJ1 and OBJ2 such that OBJ1 precedes OBJ2 in the time sequence and OBJ1 is a red bird and OBJ2 is a white bird.

```
SELECT object
CLUSTER * ALIAS OBJ1 OBJ2
FROM
{
    // it extracts all the video frames
    SELECT t
    CLUSTER *
    FROM video_source
}
WHERE OBJ1.type="bird" AND OBJ1.color="red"
AND OBJ2.type="bird" AND OBJ2.color="white"
AND OBJ1.t < OBJ2.t
```

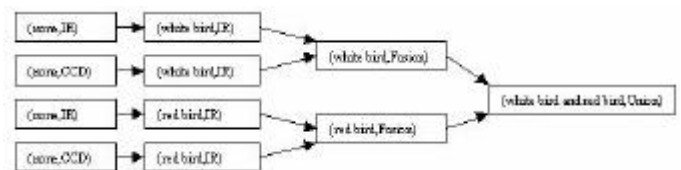
This  $\Sigma$ QL query is parsed to generate the sensor dependency tree T7 as shown below (with detailed parameters *recog\_alg<sub>i</sub>*, *time<sub>i</sub>*, *recog\_cr<sub>i</sub>*, *norecog\_cr<sub>i</sub>* omitted), which can then be processed by the query processing algorithm.



Suppose the query is:

**Query 2:** "Find a white bird that comes after a red bird."

The  $\Sigma$ QL query is identical to the  $\Sigma$ QL query for Query 1 except that the word 'video\_source' is replaced by the keyword 'any\_source'. Suppose we have two sources IR and CCD, we can construct the sensor dependency tree T8 (again omitting the other parameters) as shown below:



By checking the ontological knowledge base and the object views, the sensors and the recognition algorithms are selected. In this case, we choose (IR, algorithm123), (CCD, algorithm456) and fusion algorithm789 for the white bird, and (IR, algorithm11), (CCD, algorithm22) and fusion algorithm33 for the red bird. The tree T8 can be transformed into the corresponding  $\Sigma$ QL query as shown below. In other words, the original  $\Sigma$ QL query created by the user is now expanded into a detailed  $\Sigma$ QL query created by the query interpreter.

```

1.UNION (
2.    FUSION
3.    {
4.    SELECT object
    CLUSTER * ALIAS OBJ1
5.        FROM
6.        {
7.            // it extracts all the IR frames
8.            SELECT t
9.            CLUSTER *
10.           FROM IR
11.        }
12.    WHERE OBJ1.type= "bird" AND
    OBJ1.color = "red"
13.    SELECT object // select all frames that
    contains red birds.
14.    CLUSTER * ALIAS OBJ1
15.    FROM
16.    {
17.        // it extracts all the CCD frames
18.        SELECT t
19.        CLUSTER *
20.        FROM CCD
21.    }
22.    WHERE OBJ1.type= "bird" AND
    OBJ1.color = "red"
23.    }
24.    FUSION
25.    {
26.    SELECT object // select all frames that
    contains white birds
27.    CLUSTER * ALIAS OBJ2
28.    FROM
29.    {
30.        // it extracts all the IR frames
31.        SELECT t
32.        CLUSTER *
33.        FROM IR
34.    }
35.    OBJ2.type = "bird" AND OBJ2.color =

36.    SELECT object // select all frames that contains
    white birds
37.    CLUSTER * ALIAS OBJ2
38.    FROM
39.    {
40.        // it extracts all the CCD frames
41.        SELECT t
42.        CLUSTER *
43.        FROM CCD
44.    }
45.    OBJ2.type = "bird" AND OBJ2.color = "white"
46.    }
47.    )
48.    WHERE OBJ1.t < OBJ2.t

```

We now explain the query processing steps. Pairs of leaf nodes are selected, and the information from all the frames in order of time is put into the multi-level view database. The corresponding  $\Sigma$ QL clauses can be deleted. In this case, lines 7,8,9,10 and 17, 18, 19,20 and 30,31,32,33 and 40, 41,42,43 will be deleted.

The sensor dependency tree is now modified. In this case, the IR and CCD nodes are deleted. We can then select the next node(s) to process. We have four nodes to choose, (White Bird, IR, recog123, 10,

recog\_cr, norecog\_cr), (White Bird, CCD, recog456, 20, recog\_cr, norecog\_cr), (Red Bird, IR, recog11, 30, recog\_cr, norecog\_cr), (Red Bird, CCD, recog22, 30, recog\_cr, norecog\_cr). Since the (IR, White bird) has the smallest estimated processing time, it is selected next and algorithm123 is applied. This information (all white birds detected by IR and specified *space of interest* such as soil) are put into the multi-level view database. The query clauses (in this case, lines 26, 27, 28, 35) are deleted, and the corresponding node are deleted from the sensor dependency tree.

Since the information we get from IR doesn't fully describe an object, we need to use CCD, choose the node (CCD, White bird) and apply algorithm456 on specified *space of interest* soil. The information is put again into the multi-level database. The query clauses (lines 36,37,38,45) are deleted, and the corresponding node is deleted from the sensor dependency tree. Similar processing can be applied to the query portion dealing with red bird.

To do the fusion of white bird and red bird, we get the information about white bird and red bird with higher certainty value, again put the information into the multi-level database, delete the clauses (lines 2,3,23, and 24,25,47) and delete the nodes for fusions from the sensory dependency tree. Now only the union-node is left. We select it and modify the information in the multi-level view database, delete the query clauses (lines 1, 47, 48) and delete the node from sensor dependency tree. Since no node is left in the sensor dependency tree, we terminate the query processing.

## 5. The Optimization Problem

In the previous sections we explained the query processing steps. In this section the optimization problem will be investigated. Suppose that we have the sensor dependency tree T1 shown in Section 2.

For recog315, we have the following certainty range:

$$P(\text{recog315} = \text{yes} \mid X = \text{truck}, Y = \text{LR}) \in (0.3, 0.5).$$

$$P(\text{recog315} = \text{no} \mid X \neq \text{truck}, Y = \text{LR}) \in (0.4, 0.6).$$

where  $X = \text{truck}$ ,  $Y = \text{LR}$  means that there is a truck in the frame which is obtained by LR. To avoid interval arithmetic, in the computation of certainty values, we will use the average of the min and max of a certainty interval as its real value.

Given the sensor dependency tree, we want to recognize the object 'truck' with the certainty threshold. Our goal is to minimize the total processing time under the condition that the object 'truck' that we recognized is with higher possibility than threshold. In other words, the problem is as follows:

$$\text{Minimize } \sum_{j=1}^N \sum_{i=1}^N \mathbf{d}_{ij} T_i$$

$$\text{where } \mathbf{d}_{ij} = \begin{cases} 1 & \text{if algorithm } i \text{ runs at } j\text{th order} \\ 0 & \text{if algorithm } i \text{ doesn't run at } j\text{th order} \end{cases}$$

$$T_i = \text{processing time of algorithm } i.$$

subject to

$$\sum_{i=1}^N \mathbf{d}_{ij} \leq 1 \quad (\text{for } j\text{th order, at most one algorithm can run.})$$

$$\sum_{j=1}^N \mathbf{d}_{ij} \leq 1 \quad (\text{for every algorithm, it can be at most in one order.})$$

$$\max(\mathbf{C}^N \cdot \mathbf{d}_N) \geq \theta \quad (\theta \text{ is the certainty threshold.})$$

where

$$\mathbf{C}^1 = (c(\text{ALG}_1, \text{priori certainty}), \dots, c(\text{ALG}_N, \text{priori certainty}))$$

$$\mathbf{d}_1 = (\mathbf{d}_{11}, \dots, \mathbf{d}_{N1}).$$

$$\mathbf{C}^2 = (c(\text{ALG}_1, \mathbf{C}^1 \cdot \mathbf{d}_1), \dots, c(\text{ALG}_N, \mathbf{C}^1 \cdot \mathbf{d}_1))$$

$$\mathbf{d}_2 = (\mathbf{d}_{12}, \dots, \mathbf{d}_{N2}).$$

...

Given the sensor dependency tree, a *dual problem* is to recognize the object 'truck' within the processing time limit. Our goal is to maximize the certainty value for the object truck under the condition that the total processing time is below the time limit. The problem is as follows:

$$\text{Maximize } \max(\mathbf{C}^N \cdot \mathbf{d}_N)$$

$$\text{where } \mathbf{d}_{ij} = \begin{cases} 1 & \text{if algorithm } i \text{ runs at } j\text{th order} \\ 0 & \text{if algorithm } i \text{ doesn't run at } j\text{th order} \end{cases}$$

$$\mathbf{C}^1 = (c(\text{ALG}_1, \text{priori certainty}), \dots, c(\text{ALG}_N, \text{priori certainty}))$$

$$\mathbf{d}_1 = (\mathbf{d}_{11}, \dots, \mathbf{d}_{N1}).$$

$$\mathbf{C}^2 = (c(\text{ALG}_1, \mathbf{C}^1 \cdot \mathbf{d}_1), \dots, c(\text{ALG}_N, \mathbf{C}^1 \cdot \mathbf{d}_1))$$

$$\mathbf{d}_2 = (\mathbf{d}_{12}, \dots, \mathbf{d}_{N2}).$$

...

subject to

$$\sum_{i=1}^N \mathbf{d}_{ij} \leq 1 \quad (\text{for } j\text{th order, at most one algorithm can run.})$$

$$\sum_{j=1}^N \mathbf{d}_{ij} \leq 1 \quad (\text{for every algorithm, it can be at most in one order.})$$

$$\sum_{j=1}^N \sum_{i=1}^N \mathbf{d}_{ij} T_i \leq T \quad (T \text{ is the maximum time that we can bear.})$$

where

$$T_i = \text{processing time of algorithm } i.$$

## 6. An Experimental Prototype

An experimental prototype for query processing, fusion and visualization has been implemented. As shown in Figure 8, after a recognition algorithm is applied and some objects identified, these objects can be displayed. Each object has seven attributes: object id, object color, object type, source, recognition algorithm, estimated processing time and certainty range for object recognition. There are also hidden attributes including the parameters for the minimum enclosing rectangle of that object, the spatial and temporal coordinates of the query originator, the space of interest and time of interest, and the certainty range for non-recognition of object.

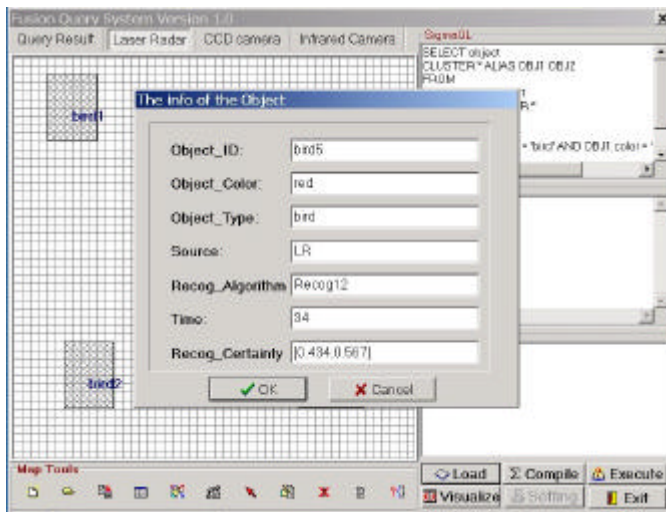


Figure 1. Objects recognized by the recognition algorithms have seven attributes.

The main window in Figure 2 illustrates the visual construction of a query. The user drags and drops objects and enters their attributes, and the constructed query is shown in the upper right window. The objects in the dependency tree are shown as an *object stream* in the middle right window. The lower right window shows the query results. When an evolutionary query is being executed, its dependency tree will change dynamically. Figure 3 displays the same information as that of the object stream, but in a format more familiar to the end users. It shows the dependency tree on the left side of the screen, and the selected node with its attributes on the right side of the screen.

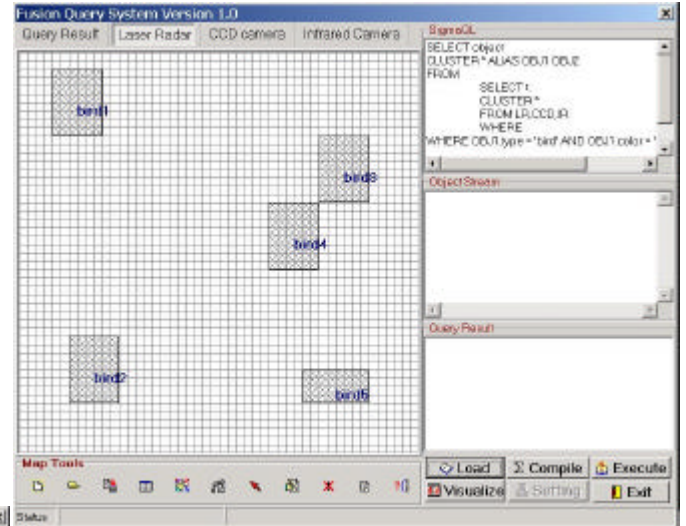


Figure 2. Visual construction of a query. The resultant query is shown in the upper right window.

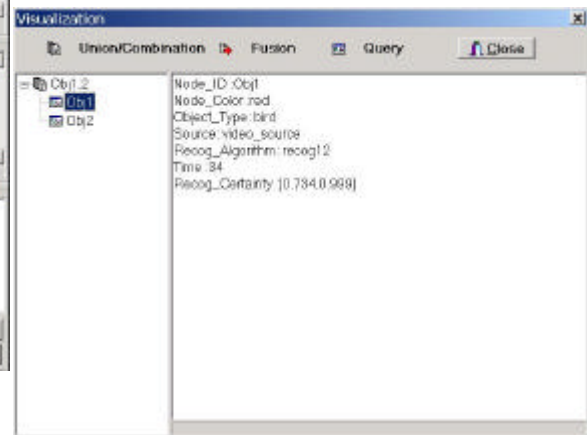


Figure 3. The dependency tree (left screen) and a selected node (right screen).

The query shown in the upper right window of Figure 2 is:

```
SELECT object
CLUSTER * ALIAS OBJ1 OBJ2
FROM
    SELECT t
    CLUSTER *
    FROM video_source
WHERE OBJ1.type = 'bird' AND OBJ1.color = 'red' AND
OBJ2.type = 'bird' AND OBJ2.color = 'white' AND OBJ1.t <
OBJ2.t
```

The corresponding Object Stream is:

```
COMBINE
OBJ1,OBJ2
OBJ1.t=OBJ2.t
```

```
OBJ2
video_source
OBJ2.type = 'bird'
OBJ2.color = 'white'
```

```
OBJ1
video_source
OBJ1.type = 'bird'
OBJ1.color = 'red'
```

The Result Set is:

```
Bird1, Bird3
Bird2, Bird4
```

i.e., either {Bird1, Bird3} or {Bird2, Bird4} is the retrieved result.

Another example is a fusion query:

```
SELECT object
CLUSTER * ALIAS OBJ1 OBJ2
FROM
  SELECT t
  CLUSTER *
  FROM LR, CCD, IR
  WHERE
WHERE OBJ1.type = 'bird' AND OBJ1.color = 'red' AND
OBJ2.type = 'bird' AND OBJ2.color = 'white' AND OBJ1.t <
OBJ2.t
```

The Object Stream is:

```
COMBINE
OBJ1,OBJ2
OBJ1.t<OBJ2.t

FUSION
OBJ2
LR, CCD, IR
OBJ2.type = 'bird'
OBJ2.color = 'white'

FUSION
OBJ1
LR, CCD, IR
OBJ1.type = 'bird'
OBJ1.color = 'red'
```

## 7. Evolutionary Query Processing, Visualization and Evaluation

As mentioned above, the data from the sensors must be merged to produce a coherent response to user queries. However given that each sensor is contributing incomplete and potentially conflicting information, it is likely that the system's response will still contain an element of uncertainty. This motivates the need for a system that can effectively display potentially ambiguous results to the viewer in a meaningful way. While the viewer is interacting

with such a display, it also makes sense for the system to interpret the viewer's actions in an attempt to resolve some of the uncertainty and adjust the presentation to reflect the clearer focus.

Under the guise of routing for emergency rescue in catastrophic events, a system could be implemented that obeys the following stages. First, a query is issued that activates the appropriate sensors to collect information about the environment. A central processor then collects data from the sensors and fuses the information into a coherent statement about the environment. The relevant information is passed to a display that helps the viewer visualize the results. Ultimately, an interaction loop between the viewer and the display allow the viewer to provide feedback and refine the query.

At the broadest level, the proposed system is fairly simple, consisting of three main interface components: a query mechanism, a visualization display, and a feedback mechanism. This general model offers the broadest possible solution and probably describes many visual information systems. Additional requirements may include:

- To process the query requires analysis of partial, ambiguous, redundant, and possibly conflicting information. Resolution/Fusion of the sensor data ensures a level of uncertainty that needs to be expressed in the visualization.
- The visualization needs to be able to assist the viewer with the discriminating relevant information from background noise.
- The evaluation of the system by the viewer needs to permit the viewer to provide feedback on the accuracy of the sensors (a), as well as the accuracy of guidance provided by the visualization (b).

To address these issues, we propose a modular approach, selecting specific technologies that address the needs for each of the components. The foundation for the Query module can rely on ÓQL and the query refinement fusion algorithms laid out in [Chang02] and discussed above. One approach to guided visualization is known as *Attentive Navigation*, based on a framework provided by [Hanson97]. A brief discussion of this method as well as a proposed feedback approach is reviewed in this section. Figure 4 shows how the components fit together powered by the suggested technology.

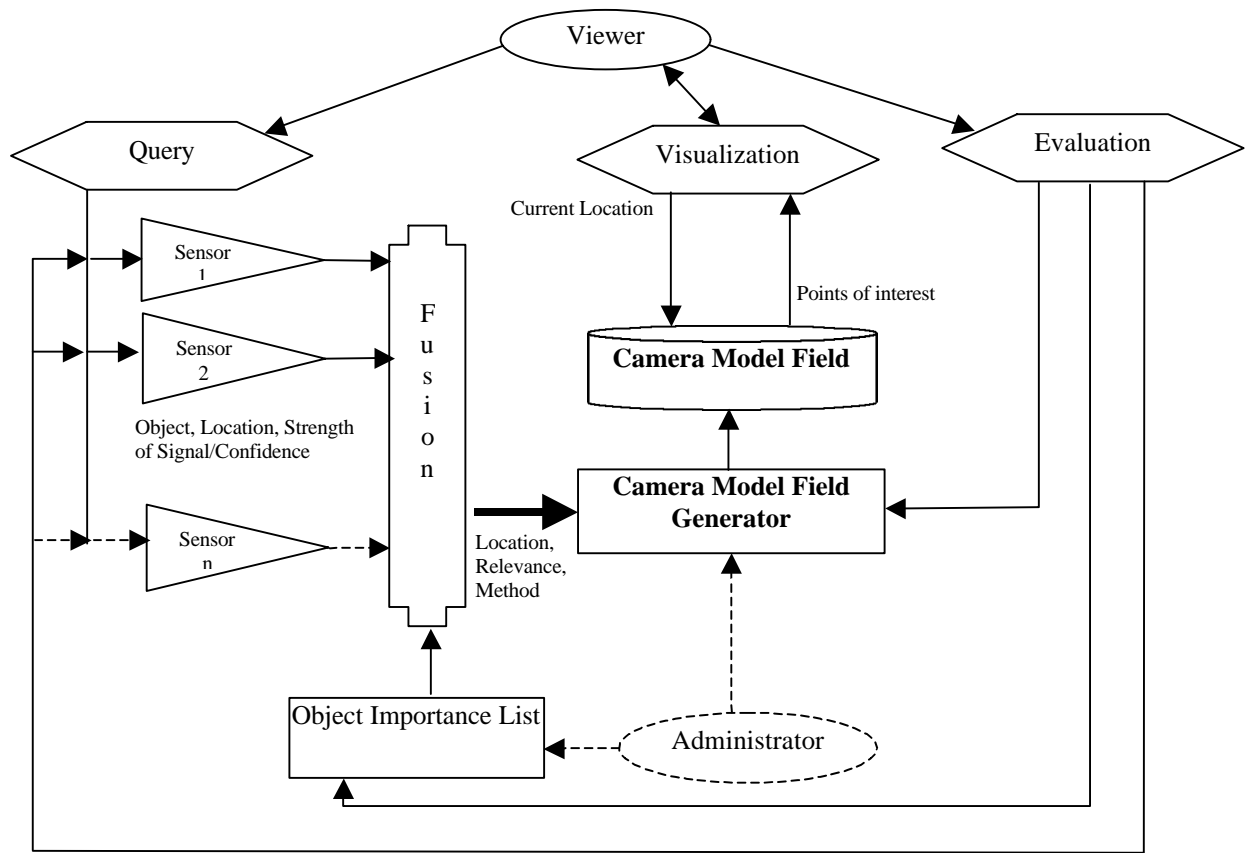


Figure 4.. Model for Query, Display and Evaluation of Dynamic Environments.

Given the uncertainty that is inherent to a dynamic environment, coupled with incompleteness introduced by the sensors, it would seem reasonable for the visualization to display a lot of contextual information. This approach allows for all known information to be displayed, while the system attempts to direct attention to key features. If a key piece of information is overlooked because of sensor uncertainty, the human-in-the-loop viewer still has a chance to rectify the problem.

Attentive Navigation is one approach to interactive 3D visualization that seems well suited to dealing with high levels of uncertainty. As the viewer adjusts the location of an egocentric viewpoint to explore the environment, the system is capable of suggesting ideal viewing parameters, based on the current position. These parameters are stored in a database known as the Camera Model Field (CMF).

Practically speaking, the CMF is not explicitly represented; as the number of viewpoint locations can be arbitrarily large. Hanson and Wernert proposed

sampling the CMF at a fixed resolution and then ensuring smooth transitions by interpolating the viewing parameters using a bicubic Catmull-Rom spline [Hanson97, Hanson99].

Different effects can be accomplished by changing the way the ideal viewing information is used during viewer interaction. An extensive taxonomy of these interaction techniques can be found in [Wernert99]. Additionally, user evaluation has been performed on some of these variations [Hughes2000, Hughes2002]. Initial findings support the using the ideal viewing vector to provide annotations in the display that direct viewers where to look. Any type of representation is possible, depending on the aesthetic requirements of the designer. Several possibilities are outlined below.

- a) *Projected Vector*: The most basic annotation that can be realized is to simply project the ideal viewing vector on the floor of the environment. This will allow the viewer to get a sense of a direction that is given to them. This may be used to tell them either the direction they should be looking or perhaps a direction they should

travel. This approach is represented in Figure 5(a).

- b) *Attentive Flashlight*: Another approach is to use the ideal viewing vector to control the direction of a spotlight traveling through the scene. With this condition, as the viewer moves through the environment, the flashlight will shine on relevant objects, leaving unimportant features in the dark. This technique may be more effective if the CMF is used to fixate on certain objects, as opposed to showing directions. An example of this treatment is shown in Figure 5(b).
- c) *Guide Avatar*: In this scenario, an animated tour guide accompanies the viewer through the scene, as shown in Figure 5(c). The avatar's body orientation is controlled by the ideal viewing vector. If the viewer needs assistance, they can direct their attention to the avatar and interpret what the avatar sees as interesting.

To support the evolutionary queries, the CMF must be able to be dynamically adjusted to reflect the current interests of the viewer, and the sensor knowledge. A generation tool is currently being developed to automate the construction of the CMF. Ideal viewing vectors are added to the CMF using three basic controls shown in Figure 6: point attraction, point repulsion, and directed area. A simple protocol integrates the CMF generator with the results of a query. Figure 4 shows the fusion output passing the triad  $\langle \text{Location, Relevance, Method} \rangle$  to the CMF generator. For example, after identifying a toxic spill, the fusion module might issue the command  $\langle (X, Y, Z), .8, \text{Repulse} \rangle$  to indicate that emergency response teams should avoid the area surrounding  $(X, Y, Z)$ . The relevance value (.8) is a combination of factors, such as object importance, degree of query match and sensor certainty. The CMF generator uses this value to establish the area affected by the command as well as the magnitude of the viewing vectors.

## 8. Discussion

This paper provides a general and formal framework for formulation and optimization of sensor-based queries, which has three major advantages: a) the ability to utilize incomplete information in sensor-based query processing and optimization, b) the generality to deal with spatial/temporal queries, and c) the achievement of sensor data independence.

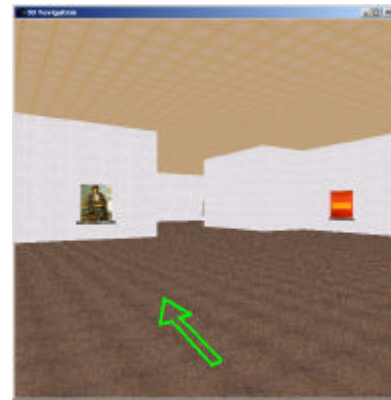


Fig 5(a)



Fig 5(b)



Fig 5(c)

Figure 5. Annotation powered by Attentive Navigation.

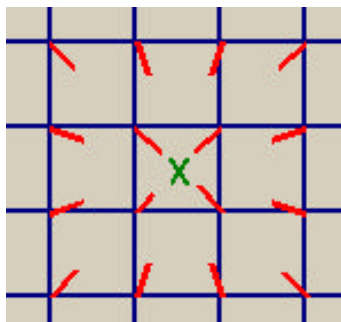


Fig. 6(a)

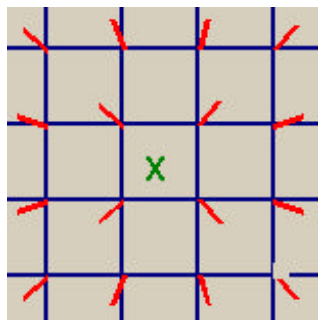


Fig. 6(b)

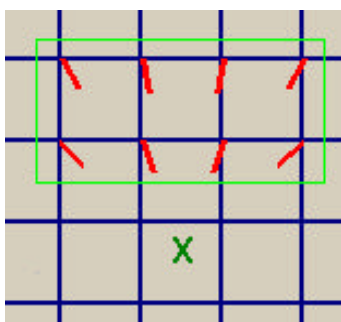


Fig. 6(c)

Figure 6. (a) Point Attraction (b) Point Repulsion (c) Directed Area.

A framework for evolutionary query, visualization and evaluation of dynamic environments is proposed. To close the loop in the system requires that the viewer be able to provide feedback, evaluating both the query results as well as the visualization itself. We are currently investigating evolutionary query optimization techniques, and various feedback approaches for query result evaluation.

## References:

[Chang98] S. K. Chang and E. Jungert, "A Spatial/temporal query language for multiple data sources in a heterogeneous information system

environment", *The International Journal of Cooperative Information Systems (IJCIS)*, vol. 7, Nos 2 & 3, 1998, pp 167-186.

[Chang99] S. K. Chang, G. Costagliola and E. Jungert, "Querying Multimedia Data Sources and Databases", *Proceedings of the 3<sup>rd</sup> International Conference on Visual Information Systems (Visual'99)*, Amsterdam, The Netherlands, June 2-4, 1999.

[Chang02] S. K. Chang, E. Jungert and G. Costagliola, "Multi-sensor Information Fusion by Query Refinement", *Proc. of 5th Int'l Conference on Visual Information Systems*, Hsin Chu, Taiwan, March 2002, pp. 1-11.

[Hanson97] Hanson, A. and E. Wernert. "Constrained 3D Navigation with 2D controllers", *Proceedings of Visualization '97*, IEEE Computer Society Press, 1997, pp. 175-182

[Hanson99] Hanson, A., E. Wernert, and S. Hughes. "Constrained navigation environments". *Scientific Visualization: Dagstuhl '97 Proceedings*, IEEE Computer Society Press, 1999, pp 95-104

[Hughes2000] Hughes, S. and M. Lewis. "Attentive Camera Navigation in Virtual Environments", *IEEE International Conference on Systems, Man & Cybernetics*. Nashville, TN, 2000, pp 967-970

[Hughes2002] Hughes, S. and M. Lewis. "Attentive Interaction Techniques for Searching Virtual Environments", *Proceedings of the Human Factors and Ergonomics Society's 46th Annual Meeting*, Baltimore MD, 2002

[Jarke84] M. Jarke and J. Cohen, "Query Optimization in Database Systems", *ACM Computing Surveys*, Vol. 16, No. 2, 1984.

[Nakashima02] Hideyuki Nakashima, "Cyber Assist Project for Situated Human Support", *Proceedings of 2002 International Conference on Distributed Multimedia Systems*, Hotel Sofitel, San Francisco Bay, September 26-28, 2002.

[Wernert99] Wernert, E. and A. Hanson, "A Framework for Assisted Exploration with Collaboration", *Proceedings of Visualization '99*, IEEE Computer Society Press. 1999, pp. 241-248.