

# Steering via the Image in Local, Distributed and Collaborative Settings

Jason Wood<sup>@</sup> and Helen Wright<sup>#</sup>

<sup>@</sup>School of Computing, University of Leeds, Leeds LS2 9JT

<sup>#</sup>Department of Computer Science, University of Hull, Hull HU6 7RX

## Abstract

Computational steering is a valuable mechanism for scientific investigation in which the parameters of a running program can be altered and the results visualized immediately. In an earlier paper we put forward an architecture for computational steering that recognises visualization as both the logical output and the input route. In this paper we investigate the practicalities of applying this architecture in local, distributed and collaborative steering. The work adopts a state-transition modelling approach, keeping parameters and results together in the display in order to present information consistently. The outcome is a manageably small set of states that could apply to many different simulations. We also present the results of a preliminary user study on the demonstrator we have constructed.

## 1. Introduction

Computational steering is often viewed in terms of a pipeline that accepts parameters at one end and produces a visualization of results at the other, with no intrinsic connection between the two. In an earlier paper [1], however, we put forward an alternative architecture for computational steering that recognises visualization as both the logical output and the input route (figure 1).

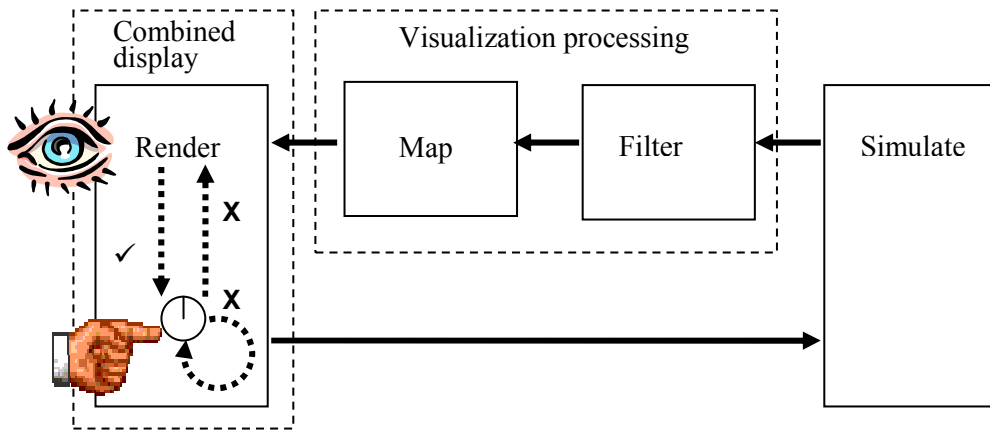
Input and output interactions with parameters and graphical results respectively are unified into one window under the control of one display process; collocation of input parameters and graphical output preserves knowledge of their respective states and in software engineering terms the architecture reflects a ‘pull’ model. A parameter object sets a new value (user interaction) and is disabled; the simulation results are marked as ‘old’; the simulation is notified that the parameter object has changed and computes new results; these new results flow to the display, re-enabling the parameter object. The benefits of this architecture are two-fold. First, the user always sees a consistent picture of their parameters and results so it is impossible to get into the confusing situation where displayed results lag many cycles behind the parameters recently input. Second, disabling the parameter object during the computation of results ensures that the steering pipeline cannot be overwhelmed by many inputs awaiting service.

In this paper we investigate how to apply this architecture in practice. We begin by examining steering in single-user and collaborative modes, gradually refining a scheme for interactions that is generic. We then go on to implement this scheme in local, distributed and collaborative settings, where two underpinning requirements are identified. First is to have the ability to support steering of simulations by multiple, remote clients. Second is to be able to manipulate parameters directly via the rendered image, using interactors in the form of geometry which can be rendered along with the results. The first requirement is furnished by modifying the gViz steering library [2], the second by modifying the multi-purpose image interaction (MPII) tools described in [3]. A description follows of the results of a preliminary user study and the paper concludes with a discussion of our findings.

## 2. Characterising Steering

### 2.1 Single-User Mode

The architecture in figure 1 deals only with the running simulation, which necessitates two states for the parameter object, namely ‘active’ and ‘inactive’. Corresponding states for the results are ‘new’ and ‘old’. Our first observation in practice, therefore, is how to deal with the ‘initial’ state of the simulation, and also the ‘paused’ state which many computational



**Figure 1.** An architecture for computational steering that collocates parameter interaction and rendering of results in a single display process. The upper arrows represent the flow of results and the lower one, parameters (from [1]).

elements allow for. In its initial state the simulation has yet to produce any results, so there is no sense in which we should mark results as ‘old’ when the parameter object is manipulated. Furthermore, there is no risk of overwhelming the pipeline at this stage, so the user should be given free rein to experiment before committing to a particular parameter value. The user should likewise have experimental freedom when the simulation is paused, but here there may well be results still on show that we have to consider. One final complication is to consider the appearance of a parameter object *during* manipulation, since in practice the switch from the active to the inactive state will not be instantaneous.

We find, therefore, that rather than simply marking results as ‘new’ or ‘old’, a more useful approach is to split the active state of the parameter object into two forms, to show if its current value is consistent or inconsistent with that used to compute the results being displayed. In order to minimise the number of states we compromise in the case of the initial simulation and define the parameter object as always being consistent when there are no results. The inactive state is unchanged in concept since it only occurs whilst the simulation is running, the case originally modelled by the architecture. In this new formulation, though, the inactive condition also of course implies inconsistency, because a re-computation is pending. Table 1 shows the minimum number of parameter object states needed to support the single-user case, together with their associated functionality. The table entries in italic font correspond to the ‘active’/‘new’ and ‘inactive’/‘old’ state-pairs originally identified in [1].

## 2.2 Collaborative Mode

Considering collaboration encourages further refinement of these states. Employing an informal floor protocol, several clients must see one another’s experiments with the parameter object. Now, a one-time-use object is not suitable since the group has to discuss and possibly try out different parameter values in the visualization, before committing just one to the simulation. At the same time, though, we wish to retain the benefits of disabling the parameter object, so must now do this with an explicit committal action, rather than on release following its manipulation.

It is interesting to note that, although this type of parameter object is only subtly different to the single-user version, to the users it changes an immediately reactive system to one where parameters are first debated and finally decided upon.

## 2.3 Managing Simulation State-Transitions

So far we have looked at the states of the parameter object and transitions between them effected by user interaction. Implicitly we have also addressed the simulation transition  $Run \rightarrow Run'$ . This is the input of new parameters during steering, served in the single-user case by manipulating and then releasing the parameter object, and in the collaborative case by a separate committal action. Other user-initiated simulation transitions must also be considered, namely  $Initial \rightarrow Run$ , and  $Paused \rightarrow Run$  and *vice versa*.

It turns out that  $Initial \rightarrow Run$  and  $Paused \rightarrow Run$  are very similar in both single-user and collaborative modes, each

Simulation state	Single-user parameter object			
	Function	State		
		Initially	During manipulation	On release
Initial	Free movement	Consistent	Consistent	Consistent
Paused	Free movement	Consistent	Inconsistent	Inconsistent
Running	Free movement, disable on release	<i>Consistent</i>	Inconsistent	<i>Inactive (Inconsistent)</i>

**Table 1.** A freely moving parameter object allows a user to experiment prior to starting the simulation, and is defined as being consistent throughout because there are no results on display. During the paused state, the initially consistent parameter object changes its state to ‘inconsistent’ to signal potential mismatch of its value with seen results. The same occurs during running, except now the parameter object can only send one value before being disabled, at the same time entering the ‘inactive’ state. Arrival of results corresponding to this parameter value re-enables this one-time-use object and resets its state to ‘consistent’.

incorporating an element of experimentation whilst the simulation is safely in suspension, prior to making a final decision to (re)commence. They are thus also similar in usage to *Run*  $\rightarrow$  *Run'* when performed in collaborative mode, so can be adequately served by the same committal mechanism.

How to treat the reverse-direction transition is more difficult to decide. Effecting a change *Run*  $\rightarrow$  *Paused* carries no risk of overwhelming the pipeline, nor does it affect the consistency or otherwise of the results that are seen. It could, therefore, be argued that this operation needs no special treatment here. A contrary view would be that *any* user interaction should be accompanied by immediate feedback. For simplicity we will adopt the former approach, providing a mechanism to make the transition but leaving the matter of feedback to the eventual cessation of results coming from the paused simulation.

Table 2 shows this new arrangement of states. The ‘On release’ outcome of table 1 has first been refined into two, namely ‘On release’ and ‘On committal’, and then subsumed into ‘During Manipulation’, with which it has become identical. Note also that the paused and running simulation states can now be treated identically. A further welcome simplification is that parameter functionality is now the same in all cases, consisting of a freely moving object that is disabled on committal.

### 3. Implementation

Table 1 supports a single user working either locally or remotely from their simulation. Table 2, the scheme we have implemented, represents collaborative steering, which by implication must be distributed for at least some of the participants. Implementing just one scheme

apparently restricts single users to a separate committal action, losing the benefits of an immediately reactive system (see 2.2). However, we have been able to resolve this limitation in practice, but with some constraints that will be described in 3.3.

#### 3.1 Modified gViz Support

The scenario envisaged in gViz [2] is of a simulation running remotely, to which multiple steering and visualization clients can connect. To support these connections, a two-part library of API routines is provided. One part, the gViz client library, deals with sending parameter values and receiving data. The other, the gViz simulation library, acts as a proxy for the simulation itself, maintaining a queue of parameters and marshalling results. Parameters are communicated between parts of the library by means of messages in XML [4].

This design required no changes to support the single-user mode described in section 2.1. In this mode, parameters chosen by the user are sent to the remote process and held in a queue within the gViz library pending delivery to the simulation. Once the simulation has taken these values and acted upon them, they are reflected back to the user representing the current state of the simulation.

If this original gViz library had been used to connect multiple steering clients, they would only have been aware of parameter changes made by a user once they had been acted upon by the simulation and become part of its current state. To support the *collaborative* multi-user mode described in section 2.2, changes were required to both components of the gViz library as well as extending the schema used to encode the XML messages. In the gViz client library, a new connection type was created called

Simulation state	Collaborative parameter object state		
	Initially	During manipulation and on release	On committal
Initial	Consistent	Consistent	Inactive
Paused/Running	Consistent	Inconsistent	Inactive

**Table 2.** In contrast with table 1, collaborative steering of a running simulation requires a committal action that is separate from manipulation and release of the parameter object, in order to allow a group of users to experiment before finally deciding on a value to send. The same mechanism can be used to put the simulation into its running state initially, and when paused. The reverse transition to pause the simulation is not treated explicitly since it does not affect the consistency or otherwise of the parameter values with seen results. Neither does it carry any risk of overwhelming the pipeline, so there is no need for a parameter that becomes inactive on use. Functionality is now the same in all cases, consisting of a freely moving parameter object that is disabled on committal, rather than on release.

steerParamCommit. This allows steering parameters to be submitted in both committed and uncommitted states; the original steerParam connection type by contrast implemented only a committed state. The gViz simulation library was similarly altered to accept and manage this new connection type. When multiple steering clients are using the steerParamCommit connection type and a user submits parameters in an uncommitted state, the library merely reflects these to all connected parties. When a user submits a set of parameters in a committed state, they are placed in the parameter queue as well as being reflected to all parties. An extra field in the XML description of the parameter carries this (un)committed state to the steering clients so they can synchronise their views of colleagues' interactions.

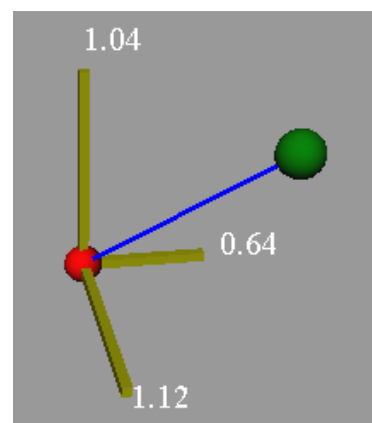
Passing parameters in this way, rather than directly client-client, maintains the star configuration of the original gViz architecture and is scalable to many participants.

### 3.2 Modified MPII Interactor

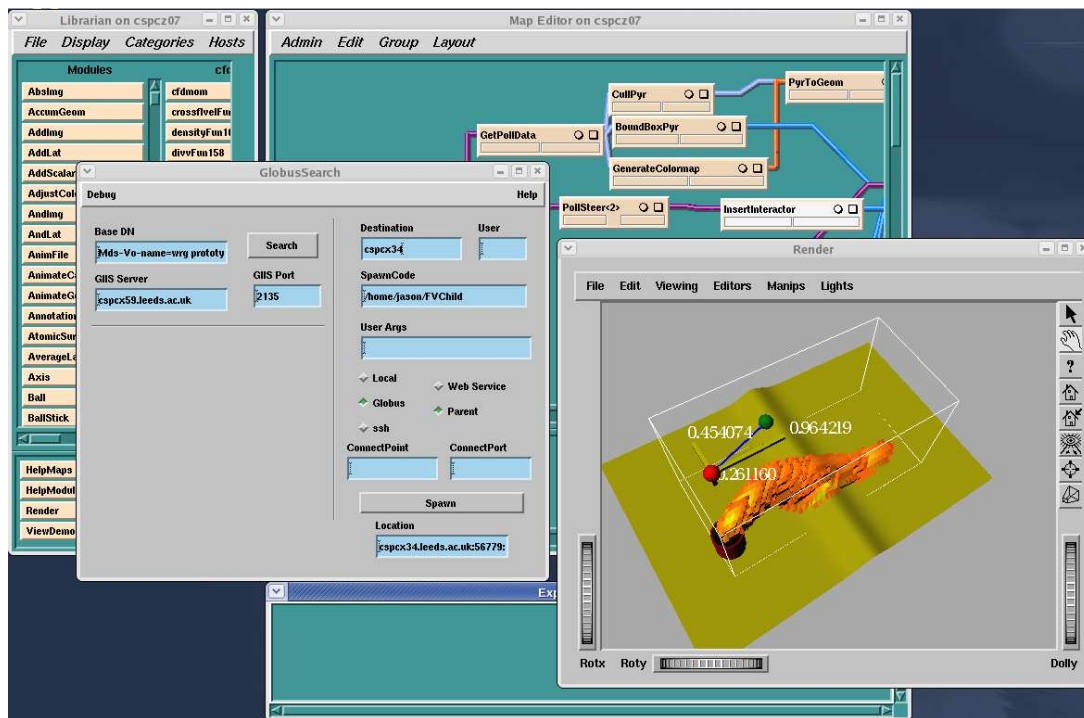
Multi-purpose image interactors [3] consist of Open Inventor [5] code that can be inserted in the visualization scene. Once added they respond to the user's mouse clicks to change their physical configuration, at the same time sending new parameter values to the simulation. These values also appear as confirmatory text on the interactor.

In this study a vector manipulator (figure 2) has been modified to incorporate knowledge of its state. To simplify matters there are two versions, one for the initial simulation state and one for its paused and running states. Since the interactor itself consists of code, it can change its own appearance when it is manipulated. This feature is used to provide feedback on its state, by changing the colour of its value text from white (consistent) to black (inconsistent) or red

(committed, see table 2). Thus version 1, used during the initial simulation state, need never change its text to black from white, whilst version 2 does so on first being moved. Once an interactor has been used to commit values its text changes to red. At the same time it constrains itself in its current position, effectively locking itself against further manipulation by the user. Repeated transmission of the same set of values is also blocked, even if the user should continue to click the interactor. In this way its appearance and lack of response cue that it is inactive – it is rendered active again once the simulation catches up and a new interactor with white text is put into the scene. The physical configuration of the new interactor matches that of the locked one it replaces, so to the user it appears that its text has simply turned white.



**Figure 2.** A vector parameter having its origin at the red sphere and pointing towards the green, with x-, y- and z-components respectively 1.12, 0.64 and 1.04. Manipulating the green sphere with the mouse changes the geometry, the text, and the compound value that is sent to the simulation.



**Figure 3.** gViz supporting image-based interaction. Local modules GetPollData and PollSteer communicate with the simulation that is started remotely by GlobusSearch. PollSteer checks the values of parameters that were used to produce each set of results and communicates with the InsertInteractor module to place the appropriate type of interactor in the scene. At the same time, GetPollData sends the results to the display.

Another issue in implementing the committal mechanism was how to distinguish a mouse click during the experimentation phase from one committing a value, solved by using the left button for ordinary movements and combined left-and-middle buttons for committals.

As well as being able to send parameter values of their own, participants must also be able to see the outcome, via their interactors, of each others' manipulations. We solve this problem using the gViz tools; all parameter values received by the simulation library, whether as a result of experimentation with the interactor or committal, are sent back to all collaborators where they are used to instantiate a new interactor in that person's scene. Rather than starting off with white text, though, (corresponding to the column labelled 'Initially' in table 2), they arrive with a colour appropriate to one of the other two columns, depending on circumstances. Conceptually we thus have one logical interactor with as many physical manifestations as there are collaborators.

### 3.3 Results

Figure 3 shows the completed demonstrator. The GlobusSearch module is used to start the simulation on a remote machine, with the remainder of the pipeline running on the local

desktop. The architecture is subtly altered from that in figure 1; here the upper and lower arrows have become decoupled because 'Simulate' is separated into two parts, one representing a connection point for the remote simulation process's parameters and the other, results. The modules PollSteer and GetPollData in figure 3 epitomise this decoupling. Now, responsibility for synchronising results and re-enabling the parameter must fall to the simulation end of the local pipeline rather than the display end. However, the effect of this change for the user is minor, with any delays at the display occurring only on account of the speed of the visualization processing, not the calculation itself.

Our experience in using the system is that an informal floor protocol works well provided participants are in direct contact by telephone or a video conference connection. If two people commit their interactors simultaneously, the gViz library arbitrarily picks one value, discarding the other. The chosen value is passed to the simulation and subsequently sent back to all users to instantiate a locked interactor. The competing user briefly sees a locked interactor with their own value that is quickly replaced by another, locked interactor representing the chosen value. The results that

then arrive are consistent with this chosen value and all interactors are unlocked.

As well as using the interactor in collaborative mode as it was intended, we have also tested it in single-user mode. Although at first sight this removes the benefits of steering an immediately reactive system (see section 2.2), the multi-button mechanism used to signal a committal can in fact be employed in this way. The combined left-and-middle buttons were originally intended to commit values when the mouse was stationary, but with practice they can be used during a drag action as well, simulating the interactor in the bottom row of table 1. It transpires that the ability to try out values before committal is useful in single-user mode too, so having both mechanisms available is an unexpected bonus.

#### 4. User Study

We have conducted a preliminary user study on the visual cues provided by the vector interactor during steering.

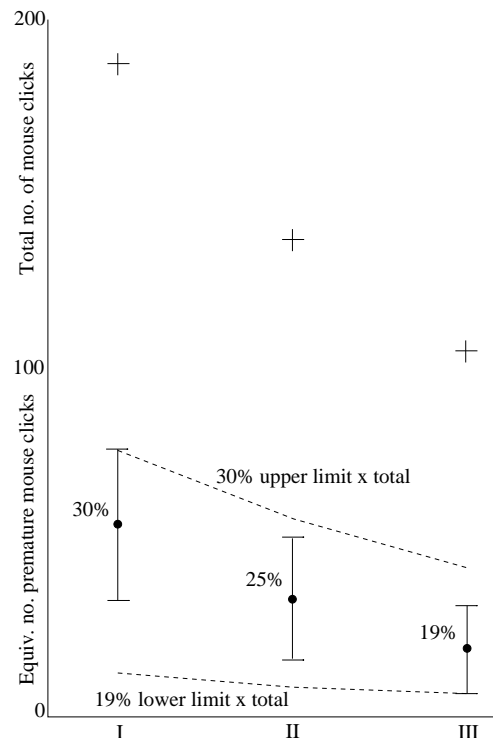
In order to test our procedure initially, two users with some prior experience of the pollution application were set the task of steering the gas plume, emitted from the smoke stack (see figure 3), towards a particular corner of the bounding box. They performed the task twice: first with an interactor whose value text remained white even whilst disabled, and second with an interactor where the text turned red. The interactor also recorded the number of 'premature' mouse clicks, that is, user events it received whilst it was in the inactive state. Both users noted unaided the different appearance of the second version, but attributed different causes. One correctly discerned that red text was associated with an inactive interactor, whilst the other assumed it signalled an error.

The interactors were then modified to use different colours – yellow throughout or green with red – in an attempt to make the meaning of the visual feedback more obvious. A further six users were tested, one of whom had some experience of the simulation and five who had none.

Each user completed three tests on a simulation generating frames of output at a little over a second apart. Between one and five frames of output might be seen between committing the parameter value and having the simulation act on it. In the first test users steered the gas plume using the yellow-numbered interactor whilst in the second they used the green-and-red version. For these two tests, instruction on visual feedback was limited

to drawing attention to the role of the numbers in confirming the sizes of the vector components, to determine whether users noticed unaided the change in colour. Before the third test, which consisted of the same task using the green-and-red interactor, users were asked if they had noticed any difference, prior to pointing out to them the significance of the numbers' colouring.

Results of the tests are plotted in figure 4, comprising the total number of mouse clicks required by all six users to complete each test, and their averaged percentages, with 95% confidence intervals, of premature clicks, expressed relative to these totals. For reasons described in the next section, all the tests I–III were carried out in the same order, so the reduction in total interactions might be attributable just to learning how to use the interface. Nonetheless, the mean percentage of clicks that were premature does fall slightly faster than this overall rate. A reduction in premature clicks in going from test I to test II would seem particularly curious since none of the users when asked had noticed any difference between these two in terms of the visual feedback the interactors offered.



**Figure 4.** The total number of mouse clicks (+) required by six users to complete the set task as visual feedback was first introduced, and then explained. The mean percentage of clicks that were premature appears to reduce slightly faster than this overall falling baseline, as shown by the dotted lines.

## 5. Discussion

Our experiences in trying to model the various forms of steering were interesting. Beginning with the single-user case we developed a scheme (table 1) that was consistent with the original aims of figure 1, but which had to be modified to accommodate several users (table 2), by incorporating a separate committal action. In theory, it appeared this collaborative interactor would be less satisfactory for a single user, but in practice this was not so. Not only did the choice of committal mechanism allow us to simulate an immediately reactive system (see 3.3), the ability for one person to experiment with parameter values without necessarily committing them each time also proved useful.

Modifying the gViz library has extended it to allow for collaborative working, in contrast to simply supporting multi-user input and output. Previously, collaboration over the steering of a simulation and the visualization of its data has been made possible through the appropriate choice of front end system. The IRIS Explorer [6] visualization environment, for example, provides a library and set of tools to support collaborative working. These extensions to the gViz library will be carried forward in the e-Viz [7] project and used to provide collaborative working capabilities for visualization and steering.

Steering a remote simulation necessitated a subtle change in the architecture in figure 1, with the ‘Simulate’ end of the pipeline now responsible for synchronisation, rather than the display end. Although not an issue for this application, potentially this introduces a delay in the rendering of results relative to the rendering of the parameter object, reducing the benefit of the visual feedback we have included. The same would apply if the visualization involved very large amounts of data, or was running on a separate machine to the steering client (see for example, [8]). Neither situation would be ideal for image-based steering; nonetheless, the principle of disabling the interactor until the simulation catches up is valuable, and one that is equally applicable to ordinary interface elements such as dials and sliders.

Our demonstrator steered a single parameter but we have given consideration to steering several within one application. There are two broad approaches: the first is to steer parameters individually, each with its own committal action. This would be a time-consuming but safe approach, since no parameter could be

committed without an explicit action from the user. The alternative is to arrange that committal of one parameter also sends all others that have changed, allowing batches of parameters to be sent. The drawback of this mechanism, however, is that parameters changed inadvertently would also be sent to the simulation. A possible compromise we intend to investigate would be to allow parameter batching at the start of steering, but then switch to sending individual values once the problem has been set up.

The preliminary user study was deliberately modest; however, though small it also provided some unexpected insights. We purposely did not randomise the test order in the belief that using the green-and-red interactor first would invalidate our conclusions about the yellow one, imagining that changes of colour on the interactors would be obvious to users without prompting. Since this is clearly not the case, running different sequences followed by a between- and within-group analysis will allow us to correct for learning effects and reveal the true contribution of visual feedback in reducing premature clicks.

Perhaps the most surprising result of all needs no further confirmation, which is simply that users continue to operate the interactor even after being instructed in respect of its inactive status. A mean of 19% premature clicks in test III represents nearly 1 in 5 opportunities for steering confusion. Independent of the perceived worth of visual feedback or of placing interactors in the image, this is surely compelling evidence of the underlying value of this architecture—constraining the user to pull results rather than push parameters.

## 6. Acknowledgements

JDW acknowledges funding under the EPSRC-supported e-Viz project (*An Advanced Environment for Enabling Visual Supercomputing*). We thank Dr Fotis Chatzinikos for the original MPII tools, developed at the University of Hull during the EPSRC project *Computational Interaction: Realising the Potential of Visualization*. Partners in the EPSRC-supported gViz project (*Visualization Middleware for e-Science*) were the Universities of Leeds, Oxford Brookes and Oxford; CLRC; NAG, IBM UK and Streamline Computing. Dr Mark Walkley (University of Leeds) was instrumental in constructing the gViz pollution demonstrator code. Grateful thanks are due to colleagues at the University of Leeds who gave generously of their time to

participate in the user study and to Dr Mike Brayshaw of the University of Hull for discussions on the data analysis.

## 7. References

- [1] Wright H, 2004, *Putting Visualization First in Computational Steering*. In: Proceedings of UK e-Science All Hands Meeting, AHM2004, Cox SJ (ed), EPSRC 2004, pp 326 – 331.
- [2] Brodlië KW, Duce DA, Gallop JR, Sagar M, Walton JPRB and Wood JD, 2004, *Visualization in Grid Computing Environments*. In: Proceedings of IEEE Visualization Conference, Vis2004, Rushmeier H, Turk G and van Wijk JJ (eds), IEEE Computer Society, pp 155 – 162.
- [3] Chatzinikos F and Wright H, 2003, *Enabling Multipurpose Image Interaction in Modular Visualization Environments*. In: Proceedings of SPIE Vol 5009, Visualization and Data Analysis 2003, Erbacher RF, Chen PC, Roberts JC, Gröhn MT and Börner K (eds), SPIE 2003, pp 203-214.
- [4] World Wide Web Consortium, 2005, *Extensible Markup Language (XML)*. <http://www.w3.org/XML/> [Accessed June 2005]
- [5] Silicon Graphics, Inc., nd, *Open Inventor*. <http://oss.sgi.com/projects/inventor/> [Accessed June 2005]
- [6] Walton JPRB, 2004, *NAG's IRIS Explorer*. In: Visualization Handbook, Johnson CR and Hansen CD (eds), Academic Press.
- [7] Riding M, Wood JD, Brodlië KW, Brooke J, Chen M, Chisnall D, Hughes C, John NW, Jones MW and Roard N, 2005, *e-Viz: Towards an Integrated Framework for High Performance Visualization*. Proceedings of UK e-Science All Hands Meeting, AHM2005, to appear.
- [8] Pickles SM, Haines R, Pinning RL and Porter AR, 2004, *Practical Tools for Computational Steering*. In: Proceedings of UK e-Science All Hands Meeting, AHM2004, Cox SJ (ed), EPSRC 2004, pp 579 – 585.