Interactions, Transformations and AUI3D

Mark D. Watson*
CMPT856

April 29, 2004

Abstract

User-interfaces have been dominated by two or two-and-a-half dimensional interfaces for the better part of the last two decades. Although much important work remains in to be done in the area of interface design and human-computer interaction with regards to two dimensions, there has been a recent, renewed interest in three-dimensional interfaces. Largely, this is due to the emergence of graphics cards capable of rendering three-dimensional scenes, rather than a great demand for interfaces of this nature. As such, there has been some difficulty in finding applications that seem to require higher dimensional interfaces. Also, there has been a great deal of difficulty in understanding how user interaction should occur within three dimensions. Presented is a value-oriented framework for capturing user interaction as a series of transformations, designed to aid the research of interactions techniques.

1 Background and Motivation

While the 70s had a great deal of user interface development, the microcomputer revolution and standardization of operating systems in the 80s brought that to a close [8]. GUIs became much more standardized and practical interface development took over.

Forms based interaction, by far the dominant interface paradigm, has evolved to a point that numerous XML languages are capable of describing their components. While there remain many unsolved issues in traditional two dimensional interfaces, the basic and traditional interaction techniques are well understood and formalized.

*mark.watson@usask.ca

Quite the opposite is true of three-dimensional interfaces. Two dimensional interfaces have always had an obvious metaphor for development; the desktop. Much of the success can been seen as a way to streamline the interactions which would have occurred on desktops, with physical papers and folders. three-dimensional interfaces do not have such a background to guide their development. Rather, three-dimensional interfaces have arisen largely because the ubiquity of powerful graphics cards.

This does not mean that three-dimensional interfaces are without any merit, but rather that there has not proven to be a "killer app" that has driven research in the field. Researchers, then, have great freedom in experimenting with interactions.

AUI3D provides a framework in which interaction techniques can be easily prototyped and explored. It does this by providing a source based description that an easily be analyzed and transformed based on user input.

1.1 The Bluewall Project

The Bluewall project began development in the spring of 2003. The Bluewall is a large display system for the development, management and analysis of computer systems. Some substantial work has been done in the area of large displays [6], although the overall goals and process of the Bluewall are somewhat different.

One major issue was the desire to create a new display environment. Complete control of a display environment has many strengths. One of the purposes of the Bluewall project is to prototype and experiment with uncommon GUI constructs, for example horseshoe scrollbars or circular menus. By freeing ourselves of a traditional GUI library, such as Swing or Cocoa, one avoids the aggravation that accompanies running into their limitations. We were distinctly aware of the limitations of the system and needed a way to extend the system as needed.

We felt that a language-based approach was the best way of dealing with the difficulty of the Bluewall's display. We then should consider the variety of languages which exist for UI design.

1.2 XML Based User Interface Languages

Since XML UI Markup Languages[1] have become so popular in the last few years, it is important to discuss them and understand their core strengths.

The vast array of XML UI languages are based around forms, and have a set of tags that allow the description of any common interface

[1]Some important projects include XAML, XIML and UIML.

component. In this manner, it is easy to describe a window filled with panels which in turn contains buttons and combo boxes. To describe this components, the system is extremely powerful, and reduce classic interfaces to a sort of HTML complexity.

However, they are not designed for experimentation and often have impoverished concepts of interaction. Which these languages can be powerful, they are clearly not right for either the Bluewall, nor any attempts to study three-dimensional interfaces.

1.3 Other Approaches

In traditional GUI libraries and widget systems we are given a set of immutable devices, like buttons and text fields. These components must be allocated and destroyed by the user, and interaction is done by a series of call to pre-constructed methods.

Zero Memory Widgets (ZMW) [5] libraries instead resemble the interaction within OpenGL[2]. There is no concept of creation or deletion of objects, and all interactions are handled via programmer functions.

In pseudo-code, we might have a function perform_operation we could give it to the system as the **window_selected(perform_operation)** such that when a window is selected, that function is immediately called. This is an incredible level of freedom for most programmers and seems intuitively good. It is a very clean system – a graphics-library styled procedural language that is based around functions.

ZWM is embedded within a procedural language, currently C. The adoption of a function-call scheme similar to ZWM-in-C seems to make perfect sense for the application. ZWM is still an extremely new technology and, while it may never become anything near a dominant UI paradigm, it is a good direction for future research to continue. However, it again is based in two dimensions, and could not be used to study three-dimensional interactions.

Ultimately, AUI3D is not based off of any of these approaches, although it does have a few core characteristics in common. It is a highly structured language like the XML UI languages, but it's core is much closer to that of a graphics system which is in turn similar to XML. While AUI3D doesn't resemble these systems explicitly, it tries to address many of the same concerns, particularly those of ZWM.

2 AUI and AUI3D

AUI3D is a description language and environment for creating user interfaces. It grows out primarily out of the work done on the AUI [9]

[2]Project home is **http://www.opengl.org/**

[10] system. As such, much of AUI3D can be understood as a migration of the concepts found in AUI to three dimensions.

2.1 AUI and PAUI

AUI was created as a programming language and environment to define plastic user interfaces. Interface Plasticity refers to the property that the functionality (and skeletal structure) of an interface is preserved, even if it is ported to multiple environments. That is, the functionality of something like a thermostat in a house with computer controller heating should be adapted to PDAs, Web Interfaces as well as the physical thermostat unit with a minimum of difficulty while maximizing utility.

The AUI expression that describes an interface is fairly small. This description is then interpreted and turned into an interface with which the user can then interact with

AUI is a functional language that also describes the interface's functionality and appearance. Although functional languages are far from a dominant paradigm, they are elegant to program in, and yield clean, clear code. AUI's syntax is clean and clear, and although it has some mark-up properties like XML, it doesn't descend into that realm of confusion.

The language, much like Haskell, views all operations as a transformation from one structure to another, such as a String to an Image in the case of opening a drawing saved to a file. This transformation model resembles the experiences of the authors in TXL [2].

The seeming biggest issue with AUI is the problem of separating the interface from the rest of the system. Surely, not all operations should be carried out within AUI, which seems to be the manner in which the system is most inclined. Still, AUI only promises a plastic interface, which is still delivered even if back end routines are written in different languages depending on the device used.

AUI recognizes that interfaces can be described in an abstract manner, such that they are not bound to any given device, and still be defined enough that they can be interpreted and run. The model it presents was both novel and simple.

PAUI [11] was an attempt to extend and recast AUI as a procedural language. The goals were: human readability, an intuitive grammar, ease in transformation, ease in transformation from a related description language to a valid expression, and the capacity for procedural programming. In this manner, it resembles ZMW.

In practice, PAUI is an imbedded language, like SQL. The language is completely free of control statements, for these features we appeal to the language in which PAUI is embedded. The reason for this design

choice is that the presence of control constructs reduces the redundancy of the language.

2.2 The Canvas and the Pin

In AUI and PAUI, every scene is composed of a canvas and some number of graphical elements (or *gels*). The canvas is the fundamental base of the display; without a canvas, no expression can be understood. When rendered, a canvas is a white box containing the graphical scene. In structural terms, a canvas is a list filled with gel-pin tuples called children. A pin is an $X-Y$ coordinate pair that gives a location on the board where the gel should be placed. In the case of a box, the box's top left corner will occupy this space. In the case of a line segment, it refers to the point of origin. Other objects have some logical method in which they are placed, usually the same as the box. When a canvas is rendered, each gel is told a location from which it should begin drawing itself. Gels are rendered in order so the last entity in the canvas list may paint over previous locations as per Painter's Algorithm.

In addition to this, PAUI also allows containment. Gels were allowed to have subcanvases that again are lists of children. Every pin uses relative coordinates, and a child can be pinned anywhere within a subcanvas. This allows the movement of a gel without the need to recalculate any child's location. This allows easier transformations, and manipulations to the canvas. This is similar to the system designed by Craighill and Fong [3].

When moving the pin-canvas metaphor to three dimensions, numerous additions to the system had to occur. Most obviously, we need to extend the pin to include a the Z coordinate, making it an $X-Y-Z$ triple. Another important aspect is the orientation of the gel. Each gel's orientation can be described in terms of a Pitch-Yaw-Roll triple. See figure 1.

We now consider the problem of containment. In two dimensions, the containment metaphor can be viewed as a "stack", each gel must rest completely on the object that contains it. In three dimensions there appear to be two containment metaphors, "on" and "inside".

"On" means that a contained gel lies on the surface of the containing gel, see figure 2. This metaphor allows AUI3D to model such interactions as buttons on panels, etc. With this metaphor, we are able to easily model all of the expressions of AUI.

"Inside" means that a contained gel is physically inside of the containing gel. This metaphor is not overly useful for interaction, but can be very useful in terms of displaying data.

AUI has a small collection of primitive gels from which all scenes are created. This list includes vectors, boxes, text, and so forth, but is not intended as an exhaustive list of graphical elements that might be

```
space<200,200>{<box<5,5>(Fill Solid),
<0,0,-15>, [45, 45, 0]>}
```

Figure 1: An AUI3D expression and the resulting scene.

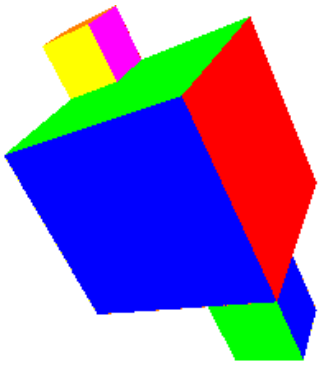


Figure 2: The two smaller boxes are both "on" faces of the larger box.

needed during GUI construction. One feature of an abstract language should the capacity to create new elements with relative ease and if it is necessary, the system is easy to extend. It is not a major undertaking to add or remove gels from the language. The parser must be modified slightly and the objects added to the interpreter, but again, this is not overly difficult. In the prototype implementation, it involves adding a method to the parser to handle the new entity and creating a class that extends the Gel class. In this manner, AUI or AUI3D can be easily extended to include whatever interface elements are required as development progresses.

2.3 The Viewer

In AUI, the relation of the scene to the viewer was static. Although the gels on the canvas could be moved to the right to give the perspective of the viewer panning left, the viewer's position *relative to the canvas* did not change.

In three dimensions, it is more important to clearly differentiate the view from the space. Where and how the viewer's perspective is defined can dramatically distort the rendering of the gels. It should not be the gels' responsibility to determine how it should be rendered given the new viewpoint. Thus, we separate transformations of the space and transformations of the view.

This requires that we add another atomic unit to the AUI expression, the **view**. As it is part of the expression, it can undergo source-to-source transformations.

Typical transformations might involve zooming, changes to the frustum, changes from objective to perspective views, and other similar changes. As they are part of the expression they can be easily manipulated to any form that the programmer wishes.

2.4 Lighting

It has been recently suggested that the inclusion of lights might be useful to the system. Lights represent another atomic element, like gels. Normally one considers a gel to be an object that the user can directly see or interact with. Lights, on the other hand, are only noticeable indirectly, that is, in term of their effect on the gels in the system.

Most graphic systems use a "three term" lighting model composed of ambient, specular and diffused lights. Specular and diffused lights are both type of lights that have physical locations, so they can be included in the expression using pins. In this regard, they may be treated as gel, although they are gels that can never contain other gels and cannot be directly interacted with.

Ambient light is general to the entire scene, and could be included as part of the description of the **space** term.

3 Transformations

At the very heart of AUI3D is a concept of source-to-source transformations. An AUI3D expression is a series of pins and gels describing some three-dimensional interface, which can be written very concisely as a string. Since we have a formal language to describe expressions, it is possible to directly manipulate them.

Source transformation is an extremely powerful paradigm [1]. We can use a transformation language like TXL to convert between two scenes. With this, it is then possible to use user input, such as a mouse click, to perform a source-to-source transformation. We will examine a few extremely simple transformations to explain the power of the system.

3.1 Movement and Rotation

The most basic transformation is the manipulation of the pins that compose the scene. This an extremely simple process – one need only need to change the value associated with the coordinates of a pin to move the object. The metaphor that AUI programmers tend to invoke is that one is "pulling the pin out of the board, and placing it somewhere else." This analogy is somewhat more complex in three dimensions, but effectively, it is the same; an object is removed and then returned, in a new location.

3.2 Insertion and Deletion

Adding and removing gels from the scene is quite simple. To add a gel to a scene, we just add its description to the expression. Similarly to destroy a gel, we need only remove its description from the expression. Clearly, these are commonly invoked transformations.

3.3 Swapping Pins

Another simple transformation which gives some example of the power of transformations is swapping pins. The point is that we switch the gels associated with two pins, which means that the gels orientations and locations will swap. Because all contained pins us relative coordinates, a gel's contents are not changed in any manner when they are transformed. For an example of swapping pins, see figure 3.

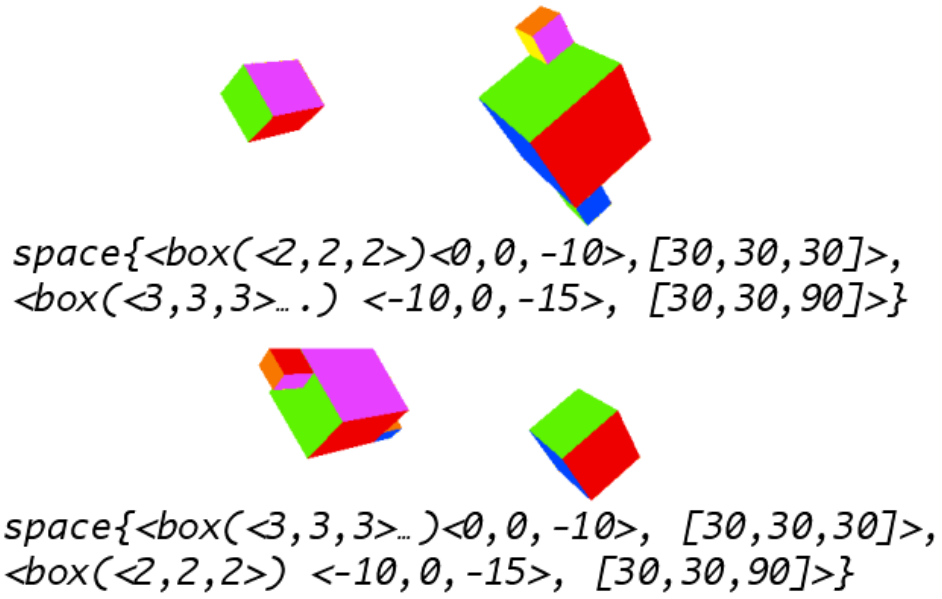
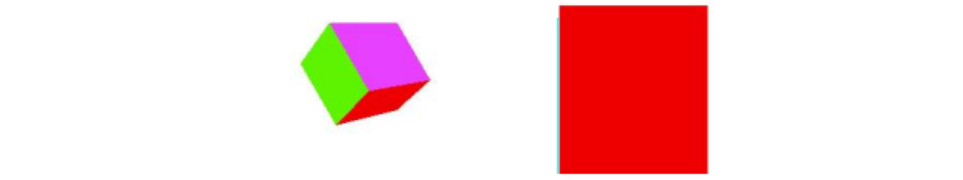
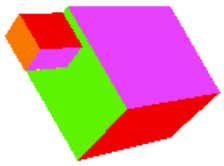


Figure 3: An example of swapping pins. The initial scene is on top, and the scene after the transformation is below. The two gels are completely unchanged, aside from the fact their orientations and locations have been exchanged.



```
space{<box(<1,1,1>)<0,0,-1>>, <box(<3,3,3>)
<-10,0,-15>, [30,30,90]>}
```



```
space{<box(<3,3,3>){<box(<1,1,1>)<0,0,-1>>}
<-10,0,-15>, [30,30,90]>}
```

Figure 4: An example of composition. The initial scene is on top, and the scene after the transformation is below. The smaller box is completely unchanged, its pin has simply been moved to be contained by the larger box. The small box appears to be huge in the first scene as it is very close to the camera.

3.4 Composition

Manipulating gels is at least as important as manipulating pins. Composition is a transformation where pins are not changed, but they are placed inside another gel. That is, one gel is transformed to contain one or more pins that it did not previously contain. Many common UI interactions can be imitated using this system, such as drag-and-drop. For an example of composition, see figure 4.

3.5 Selection

A final, common transformation is selection. Selection is a transformation based on some pattern match. For instance, we might want to change all blue squares to red ovals, or all pins beyond some X position should be rotated, or so forth. The point is that this is another simple, powerful transformation that allows wide scale manipulation to a scene.

While more complex transformations are sometimes required, it is often surprising how much is possible with very simple transformations. Often, at the core of any complex transformation is a series of trivial ones.

4 Implementation

Algorithm 1 DRAW_GELS

for all pins:

1: push a matrix
2: transform to location specified by pin's coordinates
3: rotate as specified to pin's orientation
4: tell pin's gel to **DRAW_SELF**
5: pop matrix

end

AUI3D was prototyped in C++ in both Mac Os X 10.3 as well as Windows XP. For the graphics library, OpenGL 1.5 was used.

The architecture of the system is quite simple. There are two object hierarchies: one for gels and another for views. There is another class of objects, which is the pins. There are no classes which inherit from pins.

If lights are added to the model, there seems to be a clear alteration of the model. Obviously, a **Light** element would need to be added. What should be done is that **Pin** can contain any **SceneElement**, a class that both **Gel** and **Light** inherit from. In this manner, **Pin** can be used to hold lights in positions and does not lose any of its **Gel** functionality.

The implementation makes heavy use of the keyboard as a input device to reduce the complexity of the system.

The rendering process makes heavy use of the **glMatrixPush()** and **glMatrixPop()** routines found in OpenGL. Rending is done via algorithm 1.

Note that each gel told to render itself may in turn contain gels, which are drawn using the same algorithm.

Another extension to the system would be to include interpolation between AUI3D expressions, so that rotation will appear to be gradual without needing to send numerous intermediate transformations.

5 Related Work

In the realm of creating graphics frameworks focusing on transformation, there appears to be little related works. One exception is the MAM/VRS [4], an object oriented frame work for both geometry and 3d widget design. It using a concept of nodes that define data about the objects in the system. It supports complex constraint systems and allows for animation of objects in the system. While it does possess many of the ideal features and some concept of value-oriented design, it is not explicitly based around the concept of transformation, and seems to have no idea of source-to-source transformation.

While not a framework at all, an interesting example of three-dimensional interaction for a traditional issue is 3DOSX [7]. 3DOSX is a program which replaces the Finder (window manager) in Apple's OSX operating system with a three-dimensional scene filled with rotatable platters. Each directory is represented by one of these platters with the file icons listed around its perimeter. If a user opens a folder, a new platter of that folder's contents is generated. The system also creates a connecting bridge that spans between the folder icon and it's platter. See figure 5.

While this model of interaction doesn't seem to be very effective, it is interesting that the authors are trying to reconsider a fairly well understood scenario (i.e. navigating a directory hierarchy). It is possible that this mode of thinking will reveal the scenarios in which three-dimensional interactions are radically more effective than their two dimensional counterparts.

6 Future Work

While the current implementation of AUI3D has allowed mild interaction with three-dimensional interfaces, more complex interactions must be explored. The most important extension of the work is to add functionality to the system so that we can map a traditional interface to a three-dimensional interface.

An example might be an IDE that has been mapped to a cube. On the cube, each face can have a different exclusive mode, such as different editing, compiling, debugging, and so forth. Again, it is not really known what will prove to be the benefit from moving to three dimensions, so the important thing is to experiment with the medium and see what interactions present themselves.

An important addition would be interpolation between scenes so that objects would smoothly move from one position to the next. Although this is almost purely a graphics issue, it would likely make the interface seem much more fluid and clean, and reduce the number of

Figure 5: A scene from 3DOSX. The various platters represent the contents of different folders. Folders are connected to subfolders via the bridges pictured. Taken from the developer's website.

intermediate transformations.

7 Conclusions

three-dimensional interfaces are not only an exciting new area of research for HCI researchers, they are also not very well understood. To fully explore this new domain, researchers require frameworks which are powerful enough to encompass all potential interaction techniques.

AUI3D represents a powerful framework for prototyping and experimenting with three-dimensional interactions and user interfaces. However, because of its powerful core of source-to-source transformations, a fully realized AUI3D could be used as a general interface environment. At it's core, AUI3D is a very simple system, but it can be easily extended to include any type of gel via adding classes to the interpreter.

Finally, it is interesting to note that all display aspects of AUI can be captured within AUI3D. AUI3D is a superset of AUI, and as such is capable of everything that the simpler system was capable of.

8 Acknowledgments

Thanks to the members of the CMPT856 class for providing good suggestions and to the Software Engineering Lab for the use of machines over the course of the term. Finally, thanks to Dr. David Mould and David Paquette for suggestions regarding rendering and interaction.

References

[1] Cordy, J., Dean, T., Malton, A., and Schneider, K. Source transformation in software engineering using the txl transformation system. *Journal of Information and Software Technology 44*, 13 (October 2002), 827–837.

[2] Cordy, J., Halpern, C., and Promislow, E. Txl: A rapid prototyping system for programming language dialects. In *Proceedings of the IEEE 1988 International Conference on Computer Languages* (October 1988), pp. 280–285.

[3] Craighill, N., and Fong, M. *GraphPak: A 2D Graphics Class Library*. Wiley, 1992.

[4] Döllner, J., and Hinrichs, K. Object-oriented 3d modeling, animation and interaction. *Journal of Visualization and Computer Animation 8*, 1 (1997), 33–64.

[5] Excoffier, T. Zero memory widgets. Tech. Rep. LIRIS Research Report 20030311, Université Claude Bernard, France, 2003.

[6] Guimbretière, F., Stone, M., and Winograd, T. Fluid interaction with high-resolution wall-size displays. In *Proceedings of UIST 2001 , ACM, Orlando* (November 2001).

[7] MacWarriors. 3dosx, http://www.acm.uiuc.edu/macwarriors/projects/3dosx/.

[8] Myers, B., Hudson, S. E., and Pausch, R. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction 7*, 1 (March 2000), 3–28.

[9] Schneider, K., and Cordy, J. Abstract user interfaces: a model and notation to support plasticity in interactive systems. In *Design, Specification and Verification of Interactive Systems* (2002), vol. 2220 of *Lecture Notes in Computer Science*, Springer, pp. 28–48.

[10] Schneider, K., and Cordy, J. Aui: A programming language for developing plastic interactive software. In *Proc. HICSS-35 - Hawaii Int'l Conf. on the System Sciences* (Waikoloa, Hawaii, Jan. 2002), pp. 281–291.

[11] Watson, M. Paui syntax and server: a status report. Tech. Rep. SE Lab WP03-201-MDW, University of Saskatchewan Software Engineering Lab, 2003.