Approaches to the Successful Design and Implementation of VR Applications

Steve Bryson
Computer Science Corporation/NASA Ames Research Center
Moffett Field, Ca.

1 Introduction

Virtual reality is the use of various computer graphics systems in combination with various display and interface devices to provide the effect of **immersion** in an interactive three-dimensional computer-generated environment in which the virtual objects have **spatial presence**. We call this interactive three-dimensional computer-generated environment a **virtual environment**. By immersion I mean the sense that either the user's point of view or some part of the user's body (e.g. hand) is contained *within* the computer-generated space. By presence I mean that the computer-generated objects in the virtual environment have an apparent position in three-dimensional space relative to the user.

The idea of virtual reality has generated a great deal of interest ranging from respectable research and investment to outright hype. It has become accepted wisdom, however, that the expected success of virtual reality applications has, with a few notable exceptions, largely failed materialize. There could be a variety of reasons for this failure, including the possibility that virtual reality is not as useful a medium as was originally expected. The interest in virtual reality is, however, based on the inherent three-dimensional structure of virtual reality, both in terms of display and interaction: head-tracked stereoscopic displays provide three-dimensional depth cues which are clearly superior to those provided in displays of three-dimensional environments found on conventional workstations; and six-degree-of-freedom position and orientation trackers attached to a user's hand provide an ability to control (e.g. position) object in three-dimensional space. One would certainly expect that this capability would be of great benefit in computer graphics applications which involved three-dimensional objects in three-dimensional space. Beyond these relatively mundane observations, many of the creators of virtual reality have had the vision of using the anthropomorphic character of interaction in virtual reality to create environments which, by closely mimicking human-real world interaction, create computer-generated environments which feature highly intuitive human-computer interaction. It has long been expected that such intuitive environments would facilitate user tasks by reducing the amount of "computer interaction technology" which the user would be required to master.

The vision of a highly three-dimensional environment applied to three-dimensional tasks and of highly intuitive interfaces which make the computer hardware "invisible" to the user still seems to this author an entirely reasonable and desirable vision. So what has gone wrong? Why, five years after the initial systems built at NASA Ames and the commercial systems at VPL Research, are successful virtual reality applications still very remarkable events? (I am not surprised that we are still struggling with this issue almost 30 years after the first VR system developed by Ivan Sutherland, as there are many aspects of computer science which are still catching up with Sutherland's contributions.) Looking at the history of application development in virtual reality, one failure mode becomes apparent: the available interface hardware (head-mounted displays, trackers, etc.) fails to deliver the performance required for many tasks. While this is indeed a serious problem, it is being addressed through advances in technology and will not be considered in this discussion. Another failure mode is also apparent: the failure of the design to provide the effective use of the virtual environment, either through ineffective use of the three-dimensional interface capabilities or through failing to provide the performance required to deliver the effects of immersion or presence upon which many of the benefits of virtual reality depend.

Application design which effectively provides a useful interface in virtual reality has proven to be somewhat different from design in the context of conventional three-dimensional computer graphics. Approaches to successful design for virtual reality applications is the focus of this discussion. It should be stressed that at this time there is certainly no overall theory of design for successful virtual reality applications. This presentation is meant in the spirit of a series of observations and makes no pretense at being complete.

2 Why VR Design is Different

The design process for virtual reality applications has two driving requirements:

- **The virtual environment and its interface should be tailored to the task**
- **Stringent performance constraints must be met for the benefit of virtual reality to be realized**

The first requirement is in recognition of the fact that immersing the user in a three-dimensional computer-generated environment presents many opportunities not easily found in conventional "desktop" three-dimensional graphics. Indeed much of the hype surrounding virtual reality is the recognition that one can "do anything one can imagine" in VR. While this is a highly hyped statement, it contains a grain of truth: virtual reality affords the opportunity to completely tailor the virtual environment to the task at hand. How to use this freedom effectively raises issues of overall design, design of the user interface, and important issues of human factors. The second requirement refers to the fact that the virtual environment must run with a certain minimal speed in order to be usable. Roughly, everything must happen at least ten frames per second and the system must respond to the user within a tenth of a second. These performance constraints are discussed in more detail below.

These two requirements are often in conflict: an application task (such as loading a large amount of data into memory with every visual frame, or displaying tens of millions of polygons) may be simply impossible within the 0.1 second time constraint. The successful design of a virtual reality application must simultaneously respect both of these requirements. Thus virtual reality application design is both a "top-down" (in the sense of the overall task driving the design) and a "bottom-up" (in the sense of the components and their performance driving the design) process. A good design is one that "meets in the middle", or simultaneously satisfies both design requirements.

This situation should be contrasted with the design process in conventional three-dimensional computer graphics, e.g. the design of a computer-aided design (CAD) system. In conventional three-dimensional computer graphics, the visual presentation is typically based on the Windows, Icons, Mouse and Pointer (WIMP) paradigm, in which the user is presented with a) a window which presents a view of the three-dimensional object and b) a collection of control icons (typically in another window) which the user control typically with a mouse. Manipulation of these control icons with the mouse indirectly controls the object and its view in the three-dimensional window. This is the dominant interface paradigm in conventional three-dimensional computer graphics, so the developer need not design the interface from the ground up. The performance of the application, in particular the speed at which the graphics is presented is an issue, but is not typically addressed in the design process as performance as slow as one frame per second is often considered acceptable. The optimization and tuning of the graphics performance is often done after the design is in a very advanced stage. Finally, the run-time architecture in the typical three-dimensional graphics system is based on the event-driven or callback model, in which the environment is completely passive except in response to user events such as mouse clicks.

The dominant differences between development in virtual reality and development in conventional three-dimensional computer graphics can be summarized as follows:

- **Conventional graphics has an existing dominant paradigm of interaction (the WIMP model), while VR has yet to evolve a dominant interaction paradigm.**

- **Conventional graphics design primarily considers the task, with performance considered only as an optimization phase late in the development cycle. There can be no compromise of the specification. VR cannot compromise performance so performance must be a primary consideration in the design process.**

- **Conventional graphics is typically based on an event-driven run-time architecture which assumes that user actions are the primary driver of action in the environment, and that user actions are completely unambiguous and well defined (e.g. mouse click at location). VR will often have very active environments in which objects operate independent of the user's actions. Further, user inputs will often be multi-modal and contain ambiguous and noisy information (e.g. grabbing an object with two hands better: voice and pointing).**

3 Virtual Reality Performance Constraints

The performance constraints for virtual reality call into two classes: **visual display constraint** the constraint on the display frame rate of the environment required to provide the effects of immersion and presence; and the **interactivity constraint**, the constraint on the latency time from when the user provides an input to when the system provides a response (visual or otherwise) required for the user to have useful control over objects in the environment. I wish to stress that, though these constraints are related, they are distinct. The visual display constraint refers to the frame rate of the system, while the interactivity constraint refers to the lags in the system. One constraint may be satisfied while the other may fail. Failing to satisfy the visual display constraint will lead to the failure of the illusion of immersion and presence. Failing to satisfy the interactivity constraint will lead to the inability of the user to accurately control objects in the environment through direct manipulation. There are two components to each of these constraints: a bottom-line component which will apply to **all** virtual environments, based on human factors studies; and a stricter component which will apply to objects with fast-moving objects, based on the theory of sampling.

The performance constraints based on results from human factors studies apply best to environments in which objects do not move around unless the user is manipulating them. Results from human factors studies tell us that the frame rate of the visual display must be greater than about ten frames per second in order for us to perceive changes in the visual image as continuous motion rather than as a series of still images. Note that this is *not* the frame rate required for the discreteness of the frames to be undetectable, rather this constraint is required for the cognitive effect of the interpretation of visual changes as continuous motion. The interactivity constraint is a bit more flexibly characterized: it is known that delays as small as 30 milliseconds measurably degrade human performance in tracking tasks. Humans can readily adapt, however, by slowing down the motions in the environment. If the lag time approaches 0.5 seconds, however, direct manipulation becomes very slow and essentially unusable. Experience has shown that lag times of as much as 0.1 seconds are tolerable, so long as the interaction is with an object that moves only under user control. Users can adapt to slightly longer lag, but this can seriously impact the usability of the system. It should also be mentioned that lag in VR systems can lead to severe motion sickness, though the details of this issue are not currently well understood. The bottom-line performance constraints can be summarized as follows:

For environments in which objects move only under user control:

- **(Visual display constraint) The visual images must be presented to the user with a frame rate of at least 10 frames per second.**
- **(Interactivity constraint) The lag time from when the user provides an input to when that input is reflected in the environment should be less than 0.1 seconds.**

When the virtual environment contains objects that have their own motions with which the user must interact (e.g. a virtual handball game), the user must be able to perceive enough information about the motion of the object that the user can anticipate where the object will be and how it will respond to the interaction. An example of such an interaction is the act of catching a rapidly moving ball. We can appeal to the theory of sampling to get some insight on how often the user must "sample" or be presented with the position of the object in order to anticipate its motion. Shannon's theorem from the theory of sampling tells us that an object must be sampled with a rate that is at least twice the frequency of the highest frequency of motion of the object. Shannon's theorem is a statement about our ability to reconstruct the motion of the object from the samples via a formula from fourier analysis and further assumes an infinite number of samples. Thus Shannon's theorem in only a guideline for our problem, in which the user is interpolating the object motion from a collection of samples. Experience indicates that sampling at a rate about three to four times the highest frequency of motion is sufficient for a reasonable interpolation. Sampling below this rate introduces temporal aliasing effects which can lead to very misleading perception of object motion. A similar analysis applies to the lag times allowable in the system: if the lag time is longer than the period of the sample, the visual feedback provided to the user about, for example, the user's hand position will be out of date with respect to the position of the object. There is an additional consideration based on the observation that the effect of lag on human tracking performance is frequency dependent: high frequencies of motion are more effected than lags than low frequencies. This effect leads to constraints similar to those derived from the consideration of sampling. Further, lag in applications which require rapid head motion can severely increase the risk of motion sickness. These more stringent performance requirements can be summarized as follows:

For environments which contain fast moving objects:

- **(Visual display constraint) The frame rate should be greater than three times the highest frequency of motion of the objects in the environment.**
- **(Inertactivity constraint) The lag times should not be longer than the time of a single graphics frame.**

It should be pointed out that to some extent technological improvements in, for example, the speed of computers and graphics systems will to some extent alleviate the difficulty of meeting these constraints. These technological improvements will not, however, remove these constraints as primary considerations. Experience has clearly shown that as computer power improves users expect the computers to perform more complex and demanding tasks. Thus I expect that there will, for some time to come, be virtual reality applications for which meeting these performance constraints will be a considerable challenge.

4 “Top-Down Design” as Design for the Task and the Choice of Metaphor

In this section we shall discuss some generalities about the way in which application tasks drive the "top-down" design of a virtual reality application. The particular of how an application will be implemented from the top-down will, of course, vary widely from particular application to particular application, and indeed from application domain to application domain. There are some overall themes, however, which can be discussed in the abstract. These themes are loosely encapsulated in the idea of **metaphor** in the virtual environment.

By metaphor, I mean the way in which the user is supposed to relate to the virtual environment. Metaphors are common in the theory of human-computer interaction. The idea is to allow the user to think in terms of interacting with objects which are directly related to the task at hand, rather than thinking in terms of interacting with a computer. The most common example is that of a window on a desktop containing folders as a way of organizing, interacting with, and navigating through information stored inside the computer. The window may contain folders which themselves contain information. Of course this talk of windows, desktops and folders do not literally refer to real-world windows, desktops and folders. A window on the computer screen is to be thought of as a view of an information space, metaphorically recalling the idea of a window onto another space where one currently is. Similarly the desktop on the computer screen is a metaphor for a place where folders are placed, and a folder is a metaphor for an object that contains categorized information.

In virtual reality, one has taken the fairly radical step of bringing the user into the computer-generated environment through the stress on three-dimensional immersion. This is to be contrasted with the situation in conventional desktop graphics, where the user is clearly "outside" the computer-generated environment, so the metaphor of a window into that environment is quite natural. In virtual reality, the user is inside the environment and so metaphors from conventional computer graphics such as the graphical user interface (GUI) "TV" model in which the controls are outside the environment in a separate window will be inappropriate. Thus new metaphors must be developed. We are faced with the question: "What is the metaphor for the virtual environment?"

Clearly, "bringing the user into the computer-generated environment" is itself a high-level metaphor which should be clearly of use to the application task in order to justify the use of a virtual environment for that application. Simply saying that "we shall immerse the user in the application environment" says essentially nothing, however, about the appearance of that environment to the user, what kinds of objects it contains, or how the user interacts with the environment.

Constructing a good metaphor for the virtual environment in an application is a critical task, as this metaphor will determine the appearance and behavior of the environment, as well as how the user interacts with that environment. A good metaphor will allow the user to interact comfortably and effectively with the virtual environment to perform the application tasks. A bad metaphor will obstruct the user's effectiveness, either by presenting a confusing and disorienting environment or by causing the interactions in the environment to be difficult to perform.

Constructing a metaphor for a virtual reality application is a difficult task. It is fairly clear to me that no single metaphor effectively covers all applications or application domains. Therefore the metaphors for a virtual reality application should, at this stage of maturity in the field, be constructed on a case by case basis. I shall limit this discussion to a few comments on the strategy which I have found useful in constructing metaphors for virtual reality applications.

First observe that there are several levels of metaphor in a virtual environment. I shall focus on three levels of metaphor which I feel will appear in most virtual reality applications:

- **Overall environment metaphor(s):** the metaphor which determines the overall appearance of the environment, including the types of application objects which appear in the environment. This metaphor will also impact the types of behaviors in the environment.
- **Information presentation metaphor(s):** the metaphor for how information about the environment is presented to the user.
- **Interaction metaphor(s):** the metaphor for how the user interacts with the environment and objects in the environment. This includes the overall interaction metaphor as well as individual interaction metaphors for each interactive object (such as widgets).

Note that at each level of metaphor there may be several metaphors. For example the information presentation metaphor may include text which appears in the environment (perhaps in an information window) as well as information displayed by the color of objects. The interaction metaphor may include the direct manipulation of objects as well as control via menu selection, sliders or buttons. The metaphors from one level need not match the metaphors from another level. Experience from both conventional computer graphics and virtual reality has shown that opportunistic mixing of metaphors can greatly facilitate the usefulness of virtual environments. In the virtual windtunnel, for example, there are "visualization objects" which have a behavior determined by the type of visualization they represent, and there are also "tool objects" which behave in a very different way.

Note also that the metaphors should not be assumed to derive from the real world, but should rather derive from the conventional language of the application domain for which the environment is being built. Thus, for example, an application dedicated to training for a real-world task should mimic the real-world environment of that task as closely as possible, whereas an environment for information visualization should use the language of the field from which that information derives.

One can get a handle on the choice of effective metaphors by, for each level of metaphor, asking the question "Is there a metaphor intrinsic to the application task, or rather can a metaphor be freely chosen which will optimize the task?" An example of an application with a clearly intrinsic overall environment metaphor is an architectural walk-through, where the task is to give the user the experience of being inside a building. In this application the metaphor is clearly that of being inside the building. Thus most of the visual imagery will be oriented towards presenting the building to the user from her current point of view. This application does not an intrinsic interaction metaphor for the interaction task of navigating through the building, however. While the interaction metaphor of walking may be suggested by this application, walking may not be preferred, particularly if the building is very large. Alternative metaphors for navigation could include flying though walls, pointing at where one wishes to go on a small "map" (which could be a small-scale version of the entire building) and "teleporting" there, or speaking the desired location into a voice recognition system causing teleportation. The information presentation metaphor for, e.g., one's location in the building is not even suggested by the application task. One can use the above described map, signs on the walls, make the walls transparent so that far-field navigational cues can be seen, etc.

An example of an application task in which all levels of metaphor may be intrinsic is that of training for a real-world task. In this case all aspects of the virtual environment may be required to mimic as closely as possible the real-world environment of the task for which one is being trained. In some cases, however, supplementary information may be presented to the trainee in the virtual environment during the training phase. In this case there may be opportunities for less intrinsic information presentation metaphors.

Hidden in all this talk of metaphor is an implicit issue of the human factors of interaction with a virtual environment. While there is a great deal of information available from the human factors community, much of this information is unknown to the designers of virtual reality applications. Because virtual reality is currently operating in both a far less defined user environment, takes over most if not all of the user's senses, and is very much at (or beyond) the edge of available technology, virtual reality development is much more sensitive to human factors issues than conventional computer graphics. A virtual reality interaction metaphor will not work well if it is founded upon poor human factors. Thus it is incumbent upon the designer of virtual reality applications to either become acquainted with appropriate human factors knowledge or to involve a human factors expert in the design process. Such familiarity will not address all the human factors issues, as many issues arise in virtual reality development that have not been systematically studied in the human factors community. A good working knowledge of and intuition for human factors is, however, invaluable in the virtual reality design process.

5 "Bottom-up Design" as Design for Performance

As has been stressed in previous sections, designing for the task is only part of a successful virtual reality application design process. A wonderful environment or interaction metaphor running at two frames per second will result in an unusable virtual reality system. Thus as an integral part of the design process, performance of each part and task must be considered with the same priority as how the task is to be performed. Some tasks (such as loading hundreds of megabytes or rendering several million polygons in each frame) simply cannot be done within the virtual reality performance constraints, so metaphors cannot rely on these tasks.

Designing for performance will impact several aspects of the system, including (but not limited to!):

- **The choice of the hardware platform for the system**
- **The run-time architecture of the system**
- **The choice of algorithms used in the implementation of the system**
- **The choice of data structures/representations**

etc....

In this discussion I shall concentrate on the software design issues rather on the choice of the hardware platforms to support the virtual reality system.

In the broadest terms, there are three basic classes of tasks which are in some way present in all virtual reality systems:

- **present the environment, including rendering the graphics**
- **poll the interface devices to get the current user state**
- **compute the state of the environment**

While these basic tasks are related and communicate with each other, they are functionally independent. As the graphics is to be drawn from a constantly moving point of view due to the head-tracked nature of the display, the graphics process should be decoupled from the process which computes the current state of the environment. This is analogous to the conventional drawing of a mouse cursor by a process which is independent of any particular application process which happens to be running. Similarly, as it is highly desirable for the purpose of high-accuracy interpretation of the user state, the interface devices should be polled as often as possible, keeping a recent history of the tracker state. Thus the process (or processes) which polls the interface devices should be decoupled from the graphics and computation process.

In current commercial graphics systems, each graphics engine may be accessed by only one process (though there may be more than one graphics engine, requiring more than one graphics process, in a single computer system). Thus the only run-time issue to be considered with respect to each graphics process is how the graphics is to be optimized. There are several optimization strategies, ranging from writing tight graphics code through clever culling schemes through level-of-detail rendering strategies to time-critical rendering strategies. Many of these strategies are still in the research stage. I regret that space limitations prevent me from a detailed discussion of these issues.

Based on the above discussion, it is clear that the overall run-time structure of a virtual reality application will make use of multi-processing operating systems. There will at least be a process for the graphics and a different process for the computation. The computation process may compute the geometry to be rendered (which may change with every frame), so a great deal of data may need to be communicated between these processes. It is important that the operating system which one uses supports lightweight processes ("threads"), which can communicate through shared memory. Lightweight processes have the advantage of lower overhead when the process is switched in and out, and shared-memory interprocess communication reduces the interprocess communications overhead.

The run-time architecture of the computation process is less determined. There are three models currently being considered:

- **The event-driven architecture,** in which the computation process is idle until a user event occurs, at which time a computation takes place.
- **The simulation loop architecture,** in which a program loop repeatedly executes computing the entire state of the system. Processing the user input would be one of the actions performed in this loop.
- **The fully concurrent architecture,** in which each object in the environment (including the user or users) owns its own process or processes which are continually executing asynchronously as fast as they can.

The event-driven architecture is the common model in WIMP-based conventional computer graphics. It assumes that the user is in full control of the environment and that user actions are unambiguously defined both in terms of their meaning and the temporal order in which they occur. These assumptions are typically not true in many virtual reality applications: many environments will be active, with behavior and computation independent of the user actions; input from the user can be from several sources and the interpretation of a user action may depend on the total of these inputs; and the simultaneity of inputs may determine their meaning. Thus in many applications a user event may be a high-level function of the "user state", which must be interpreted before the event can be identified. But this implies that some computation is continually occurring which is distinct from the event-driven model. Finally, the metaphor of the event-driven model, in which the user is "in control" of the environment, does not mesh well with the metaphor of immersion. In an immersive virtual reality system the user has "stepped into" the application environment, suggesting a metaphor of the user being equal (or, as Andy van Dam has put it, first among equals) to other objects in the environment. For these reasons the event-driven architecture is appropriate only for a limited class of virtual environment applications. Environments for which the event-driven model would be appropriate would be passive except in response to user events, and would have highly limited user input capabilities. An example of such an environment would include a simple architectural walk-through in which the environment is a completely passive 3D model of a building and the user interaction is limited to a single glove and tracker.

The simulation loop architecture moves much closer to the model in which the user and virtual objects are treated in essentially the same way, as tasks handled by the simulation loop. Simulation loops have the advantage of a strong sense of time flow and synchronization: a step in time is defined as one pass through the loop, and everything that happens in a single loop is defined as simultaneous. The main disadvantage of a simulation loop is that the time to perform the loop will be limited by the time to perform the slowest operation in that loop: even fast operations will be limited by the slow ones. In environments where time synchronicity is not an important consideration this may be a highly undesirable feature. Parallelizing the simulation when many processors are available may be a partial solution to this problem. Load-balancing the overall computation in a parallelized simulation loop can be an issue, however.

The fully concurrent architecture assigns a process (or a group of processes) to each object in the virtual environment. ("Object" may refer to an abstract structure in the environment rather than a single graphics object.) This architecture has the advantage that an object whose computation is fast will not be limited in performance by an object whose computation is slow. This architecture has the disadvantage that objects whose computations complete on very different time scales (i.e. one object's state computation may be 100 time faster than another's) will be rendered in states which are valid at different times and are therefore out of sync. Applications which are appropriate for a concurrent architecture are those which contain objects whose computations are very fast and comparable in speed to each other. An example may include an n-body gravity simulator, in which each object is assigned its own computational process. Sadly, another disadvantage of concurrent architectures is that many current operating systems do not support real-time handling of many processes with the time granularity and regularity required to support a smoothly running virtual environment. Hopefully this situation will improve in the near future.

Like designing the graphics for performance, the choice of computational algorithm should be driven by performance. This problem is somewhat different from classical real-time computing, in which a task must complete within a specified time. In classical real-time systems, programs are typically hard-coded to guarantee that the longest task will complete within the required time. In virtual reality systems, the tasks are generated by the user so that it is in general difficult to predict what code will be competing for computational resources. Thus hard-coded strategies will typically be too limiting for most virtual reality applications. There are various strategies for enhancing computational performance: writing optimized code; choosing computational algorithms which are faster but perhaps more inaccurate; relegating time-consuming computations to occasional, user triggered actions (experience indicates that a user will tolerate long pauses in a system so long as the user understands that is what will happen in response to a command); parameterizing the computation so that it will complete within a specified time (e.g. specifying the number of times an iterative computation is supposed to occur); and time-critical computing. Time critical computing is the design of algorithms so that they may come to some sense of completion within a specified time. The choice of algorithm, parameterization of the computation, and the parameters of a time-critical computation may be specified by the user or automatically determined. Such computational strategies are an active area of research.

Similar considerations will apply to the choice of data structures and representations as well as other aspects of the computational environment. Designing nontrivial computations for performance as well as accuracy is a new and active area of computer science.

6 Reconciling the Top-down and Bottom-up Approaches

It is fairly well understood how to design an application either from the "top-down" for a particular task or from the "bottom-up" for performance. While it is sometimes easy to design from both ends at once, in general (and in particular for complex applications in a highly performance-critical application) simultaneously implementing a successful “top-down” and “bottom-up” design strategy is a highly non-trivial undertaking. It is to be expected that there will be design flaws in a new application from both ends of the design process: metaphors and interaction tasks may be implemented which do not work well for the users or cannot be implemented with sufficiently fast performance, while tasks designed for performance may not easily fit into a good metaphor. Thus a process of feedback and refinement should be built into the design and development process from the beginning.

To test the metaphors, user testing should be an integral part of the design and implementation process. User feedback from the expected user community should be sought both in response to the proposed design of the system as well as in response to actual trial implementations of the environment and its interactions. This process will also test in an early stage the particular implementations of the metaphors. If user testing shows that the metaphors are poorly chosen, new ones should be implemented and tested. One potential difficulty at this stage is that it may be difficult to tell if a poor interaction technique or metaphor is bad because the metaphor is bad from a human factors point of view or if it is bad because it is not performing with the required speed. It is thus important to be able to implement each metaphor in a context which is guaranteed to have high performance for user testing.

If performance problems are discovered, one will, of course, first try to improve the “bottom-up” design aspects of the system to improve the performance of the existing metaphor. If redesign for performance fails to deliver the performance required to make the metaphor effective, it may be necessary to significantly modify or drop that particular metaphor. This may cause it to be necessary to re-specify what tasks the application will be capable of performing. The end users should clearly be involved at this level of the design process.

The design process in conventional computer graphics is based on the “waterfall” model, in which a specification is turned into a design which is turned into an implementation. At every step the overall consideration is to meet the specification. The overall process of design for VR that arises from these considerations is one of an iterative process at several levels, as contrasted to the more common but linear “waterfall” design process. The design process should, in order to enhance the chances of success, be highly flexible with a capability to react to significant redesigns discovered in the implementation process. In order to discover the needed redesigns at an appropriate stage of the design and development process, performance analysis, testing and user testing should be an integral part of this process. Choice of metaphor will effect the choice of implementation and vice versa. The application task will determine what metaphors to choose, but the metaphors which respect the performance limitations of the system may in turn determine which application tasks the system will support. Any large virtual reality application development effort should have built into it the flexibility required to perform this type of iterative design.

7 Acknowledgments

Many of the observations made in this essay are inspired by many conversations with Andy van Dam of Brown University and Randy Pausch of the University of Virginia. These individuals share much of the credit for the insights which led to this paper. Any errors or bad ideas are entirely the responsibility of the author.

8 References

There are many references for the material in this essay. For a survey and review of many of the design issues discussed here, see “VR as a Forcing Function: Software Implications of a New Paradigm” by Andries van Dam in the proceedings of the *1993 IEEE Symposium on Research Frontiers in Virtual Reality*.

For an overview of implementation issues in virtual reality, see the series of course notes from the courses chaired by Bryson from *SIGGRAPH 92*, *93* and *94*, and *The Science of Virtual Reality and Virtual Environments* by Roy S. Kalawsky.