

Renewing *The Erl King*
Jeff Rothenberg
January 2006

1. Introduction

Artists often utilize new representational and expressive media and new technologies to produce innovative artistic effects and experiences. In recent decades, audio and visual technology including photography, cinema, and video recording have enabled artists to create multi-media works, while affordable computer technology has enabled them to endow these works with complex, interactive behavior. Using a combination of analog and digital recording and presentation techniques, custom-built hardware devices, off-the-shelf digital computers, and custom computer programs, such works have defined an entire new, multi-faceted, unconstrained genre of art that defies simple characterization. For the purposes of this report, I will refer to such works as “computer-based” art, though this rather unimaginative term does not begin to do justice to the artistic potential of the genre. A computer-based artwork is one that uses a program to control or generate a work of art.¹

Unfortunately, the reliance of computer-based artworks on the latest technology makes them vulnerable to technological obsolescence. Digital storage media—including magnetic disks, tape, and optical disks, such as CDs and DVDs—can become physically unreadable in as little as 5 to 10 years, and they may become obsolete even faster than that. Moreover, the formats in which digital information are encoded evolve quite quickly, and new software is often unable to properly interpret older formats.² Finally, software itself becomes unusable as the computer platforms on which it runs become obsolete and unmaintainable. Any one of these factors can render digital information—such as that which constitutes computer-based artworks—inaccessible.³

As a result of these factors, computer-based artworks can become unusable in as little as a decade. In an attempt to address the obsolescence of computer-based artworks of these kinds—as well as to address the temporal loss of a wide range of other art forms employing media that decay or otherwise become unmaintainable or unusable over very short time periods—the Variable Media Network⁴ was co-founded by the Guggenheim Foundation and the Daniel Langlois Foundation for Art, Science and Technology. The NEA subsequently funded the Archiving the Avant Garde

¹. In some cases, a program may do more than simply control the behavior of an artwork: it may actually generate its content. A simple example of such a program is a “screensaver” that displays random graphical images: such a program may generate these images as it runs, rather than displaying stored image files. A program of this kind does more than control a work—it *is* the work, in a fundamental sense.

². The distinction between *software* and *programs* is that software is the stuff of which computer programs are composed, in much the same way that water is the stuff of which oceans are composed. Though it is not uncommon, it is therefore just as inappropriate (or at least awkward) to refer to programs as “software programs” as it would be to refer to oceans as “water oceans”.

³. For a further discussion of the underlying sources of the vulnerability of digital information, see [6].

⁴. As described at <http://www.variablemedia.net>, the Variable Media Network is concerned with the full gamut of issues concerning ephemeral and short-lived artworks. This report focuses exclusively on computer-based artworks, which comprise a small but growing subset of the larger class of ephemeral or “variable media” works.

project to explore related issues; this is a consortium project consisting of the University of California Berkeley Art Museum and Pacific Film Archive, the Guggenheim Museum, the Cleveland Performance Art Festival and Archive, the Franklin Furnace Archive, and Rhizome.org.⁵

The Variable Media Network seeks to involve artists—or their surviving representatives—in making decisions about how their works should be preserved or restored over time.⁶ A number of alternative preservation strategies for computer-based works have been proposed, some of which apply more readily to some types of artwork than to others.⁷

One strategy for preserving computer-based artwork is to use hardware emulation to enable modern computers to run programs written for older, obsolete computers. Emulation is a well-established computer science technique that has been used for decades. In order to evaluate the potential of emulation for preserving computer-based artworks, the Variable Media Network began a project in October 2002 to explore the use of emulation to renew Roberta Friedman and Grahame Weinbren's mid-1980s interactive video artwork *The Erl King*, which was in danger of being lost to technological obsolescence.⁸ This case study was generously supported by the Daniel Langlois Foundation.

This report motivates and describes the *Erl King* renewal project. In so doing, it examines the distinction between source code and object code programs, the difference between digital preservation efforts based on emulating computer hardware and those based on interpreting source code, issues surrounding the conversion of analog content into digital form, and techniques for dealing with the fact that computer-based artworks depend on the processors and peripheral (e.g., input/output) devices of the computer platforms on which their programs run.

⁵. The *Archiving the Avant Garde* project is described at http://bampfa.berkeley.edu/about_bampfa/avantgarde.html, where a digital version of this report—as well other, related documents—can be found.

⁶. Although the terms *preservation* and *restoration* are used somewhat interchangeably in this report, they imply different perspectives. Preservation is the more prospective and general of the two terms, whereas restoration can be thought of as an aspect or implementation of preservation. When our project began, *The Erl King* was still entirely functional and was not in need of repair; after considerable discussion, we therefore decided to use the term “renewal” to refer to what we did, i.e., perform a *proactive* restoration of the work to ensure that it would remain usable in the future, despite the fact that it did not yet require it.

⁷. Many such alternative approaches to preserving computer-based material are described in [7].

⁸. Although the Variable Media initiative uses the term *emulation* in a somewhat more general sense than the way it is used in computer science, these two meanings converge in the case of preserving computer-based art.

2. *The Erl King*

The Erl King, created between 1982 and 1985 by Roberta Friedman and Grahame Weinbren, is a fascinating and beautiful work of interactive video art based on the confluence of Goethe's haunting poem *Erlkoenig*⁹ and the disturbing anecdote of the burning child in Freud's *Interpretation of Dreams*.¹⁰ The video is built around Schubert's dramatic setting of the Goethe poem in a lied.¹¹ To avoid confusion, *The Erl King* is used in this report to refer to the video work, rather than the Goethe poem or the Schubert song.

The Erl King was first installed at the Walker Art Center in Minneapolis in 1985 and has been exhibited numerous times since then, in various major museums and galleries. In a typical installation, a single "user" sits at a touch-sensitive color display screen. Video is displayed on this screen while audio is played on speakers. If the user touches the screen, the visual and/or audio content changes in response. No menus or icons are displayed, and there is no keyboard or other input device aside from the touchscreen.

The backbone of the work consists of a video presentation of a filmed 1983 performance by soprano Elizabeth Arnold (accompanied by pianist Dean Johnson) of the Schubert lied. However, if the user touches the screen, this output is modified in various ways. For example, a touch may cause video or still images of New York City street scenes to be displayed while the Schubert continues to play on the audio track, or the display may continue to show the performance of the lied while the audio track delivers bagpipe music or alternative narrative (for example, the Freudian dream analysis). In other cases both video and audio may change at the same time, multiple video images may be superimposed, or text may appear on the screen, overlaying the video imagery. These interactive responses are controlled by a computer program that was designed by the artists and written by a small programming team.

Only a single user interacts with the piece at any given time, but others can view and hear the results of that interaction by watching a second "repeater" display screen, effectively allowing a small audience to look over the user's shoulder without physically doing so. Due to technical limitations in *The Erl King's* original hardware, the repeater screen cannot show the overlaid text or graphics that may appear on the user's screen.

From an artistic perspective, *The Erl King* was highly innovative for its time, providing a human-in-the-loop interactive experience of a kind that had (at the time) only recently become available to the artistic community through the development of reasonably inexpensive personal computers and video players. *The Erl King* required fairly elaborate programming of a low-end

⁹ The meaning of Goethe's title is obscure, though it is conventionally translated as "Elf King" referring to a menacing character who represents death and threatens to steal a father's ailing son as they ride through a forest. Alternative online renditions of the poem are available at http://www.poetryconnection.net/poets/Johann_Wolfgang_von_Goethe/13804 and <http://www.moonfairye.com/library/Erlkoenig/erlking.htm>.

¹⁰ See <http://www.freud.org.uk/BURNINGBOY.HTML> or <http://psychclassics.yorku.ca/Freud/Dreams/dreams7a.htm>.

¹¹ This song, which Schubert composed at an age somewhere between 17 and 19 (depending on the source), was the first of his works to make him famous and was one of the most widely published pieces of sheet music of the 19th century. Schubert wrote six distinct settings of this lied, for various numbers and ranges of voices. The song's relentlessly percussive accompaniment is sometimes referred to by pianists as "death to the right hand".

microcomputer to control its interactions. In addition, in order to enable audio/visual content to be changed rapidly in response to user inputs, the artists had to transfer their original 16mm color film footage to analog video and use multiple laserdisc players to switch among audio and video streams.

3. The original implementation of *The Erl King*

The Erl King consisted of several components, including a combination of off-the-shelf and specially constructed electronics, a microcomputer, and custom-written software. It is important to note, however, that the artists did not feel that the original implementation of *The Erl King* was essential to its nature. That is, the particular choice of computer and input and output devices was not considered to be the essence of the work. Instead, its essence was felt to reside in its behavior and its use of its audio/visual content. In fact, all installations of *The Erl King* hid the computer and other hardware that implemented the work, so that users could be affected only by the outputs of the system. This was a crucial factor in our approach to renewing the work by replacing its original hardware with new equivalent hardware or software: we felt that doing so would not change the nature of the work, so long as the renewed version behaved the same way as the original. Nevertheless, it is necessary to understand the original hardware that was used to implement *The Erl King*, in order to understand our approach to renewing it.

The content of *The Erl King* consists of about 90 minutes of analog video and 3 hours of analog audio material, spread over three video discs. The original visual material for *The Erl King* was shot on 16mm color sound film at the standard sound film speed of 24 frames per second (fps). The audio was originally mastered in a professional film audio mix onto 4-track perforated 35mm magnetic stock. The audio and visual streams were then synchronized and transferred onto 1-inch magnetic videotape pre-masters, at the normal analog video rate of 30 fps.¹² The standard “3:2 pulldown” technique was used to convert the film to video: this process duplicates and merges selected frames of a 24 fps source to produce a 30 fps result.¹³ The pre-master videotapes were then edited down to three 30-minute 1-inch master videotapes, the contents of which were then “burned” onto glass optical laserdisc masters, from which 10 copies were pressed. Each of the resulting three optical laserdiscs was mounted on a Sony Laserdisc player (LDP).¹⁴ Each LDP also had two audio channels, and all six channels were connected to the audio output of *The Erl King*, allowing merged audio from any combination of these channels to be played over the speakers, through a simple sound system.

¹² Although the frame rate of the 525-line NTSC video typically used in the U.S. is actually 29.97 fps, it is referred to conventionally (and throughout this report) as 30 fps.

¹³ An excellent and detailed explanation of the somewhat obscure 3:2 pulldown process can be found at http://www.dvdfile.com/news/special_report/production_a_z/3_2_pulldown.htm.

¹⁴ These were Sony Lasermax players, model LDP 2000.

The Erl King's Carroll touch screen, color display and LDPs were connected to a desk-sized Sony SMC-70 computer, which controlled the work. The SMC-70 used a Zilog Z80 microprocessor.¹⁵ The SMC-70 computer included an “RGB superimposer” board that allowed multiple images to be overlaid on the display screen. It also provided two serial ports, as well as two additional control ports that were originally designed to control audio cassette tape players that served as file storage devices. *The Erl King* used the Sony’s serial ports to control the video and audio outputs of the LDPs and to get input from the touchscreen; it used the audio cassette ports to control a custom-built “video switcher” device that could switch the outputs of any of the three LDPs to the display. The SMC-70 also included a special board containing 256K bytes of dynamic memory, which was used as a virtual “cache disk” to improve display performance.¹⁶

The Sony computer ran the popular microcomputer operating system CP/M (“Control Program/Micro”), developed by Digital Research.¹⁷ *The Erl King* also made use of a custom interactive video control program, called Limosine, which was designed by Grahame and Jon Weinbren and written by the programming team.¹⁸ Limosine actually consisted of two distinct pieces: (1) an authoring system (referred to as Limosine-A) that enabled the creation of interactive video programs and (2) a runtime system (referred to as Limosine-R) that controlled the interactive video when it ran. These programs were written in a compiled version of the Pascal programming language, called Pascal MT+.¹⁹ The implications of using a compiled language are discussed in later sections.

It is interesting to note that *The Erl King's* implementation was conceptually quite similar to that of the BBC Domesday project, which was also created in the mid-1980s.²⁰ Although it is not a work of art, *per se*, the BBC Domesday project was also an interactive video work that relied on a computer that controlled video laserdisc content. The computer and programming for the BBC Domesday project were different from those used for *The Erl King*, but the architecture and audio/video behavior of the two efforts were similar, and both had similarly become nearly unusable in a mere fifteen years, due to computer and other hardware obsolescence.

¹⁵ The SMC-70, which was one of the first computers to use “floppy” disks, is described in detail at <http://www.old-computers.com/museum/computer.asp?st=1&c=362>. The Z80 chip was a direct replacement for the Intel 8080 microprocessor, which was the chip that ushered in the personal computer revolution. When it was introduced, the Z80 was twice as fast as the 8080 (2 MHz instead of 1) and had a double set of internal storage registers, but it executed the same instruction set. (The version of the Z80 used in the SMC-70 was the Z80A, whose clock speed was 4MHz.). The Z80 is described at <http://www.geocities.com/SiliconValley/Peaks/3938/z80brief.htm>.

¹⁶ This amount of memory is much larger than the 64K byte addressing capability of the Z80 processor, which explains why this board was treated as a virtual disk rather than as an extension of the main random access memory (RAM).

¹⁷ CP/M is described at <http://museum.sysun.com/museum/cpm>.

¹⁸ The programming effort was led initially by the artist’s brother, Jon Weinbren, and later by Steve Bannasch.

¹⁹ A manual for Pascal MT+ is available online at www.cpm.z80.de/manuals/pmt68kpg.pdf. By sheer coincidence, the author of this report produced an early microcomputer software product consisting of several thousand lines of code written in Pascal MT+ running under CP/M.

²⁰ The BBC Domesday project is described at <http://www.atsf.co.uk/dottext/domesday.html>. It provided a view of British culture in the mid-1980s and was developed for the 900th anniversary of the original *Domesday Book*, which was produced under William the Conqueror in 1086. Although the original *Domesday Book* is still readable after 900 years, the BBC Domesday project became unusable in a little over a decade. See also [1] and [5].

4. Preservation characteristics of *The Erl King*

The Erl King has been installed in a number of venues over the years, so prior to our renewal effort, Grahame Weinbren kept its original hardware and software in working order in his Manhattan studio. However, the hardware had long been obsolete and was becoming increasingly difficult to maintain. The Sony laserdisc players were no longer manufactured, so the artist collected spares on eBay. The laserdiscs themselves could not easily be duplicated, and any attempt to duplicate their content would degrade its quality, since copying any analog medium creates successive copy generations, each of which is of lower quality than the last.²¹ The Sony SMC-70 computer still ran, but it would be difficult or impossible to replace when it stopped working, as it would eventually. The original Carroll touch screen had become balky and no longer responded reliably, introducing an uncertainty and delay into the interaction that were not intended.

For these reasons—and in view of the work’s artistic and historical value—we chose *The Erl King* as a case study for digital preservation. The intent of this effort was to renew the work in a modern form that would arrest its aging and enable it to run on modern and future computers and display hardware. The intent was *not* to create a new, different version of the work but rather to retain the original work’s behavior as precisely as possible.²² Simply stated, this renewal effort would require reproducing the touch screen interaction of the original, to deliver the original audio/video content to a user.

In order to ensure that the interaction of the renewed version of the work would be as close as possible to that of the original, we decided to make the renewed version run the same program that controlled the original work. The importance of this point cannot be overstated. The behavior of all but the simplest of programs is notoriously difficult to recreate precisely in a new program. That is, rewriting or modifying a program in any way is virtually certain to change its behavior, even if only subtly.²³ Such changes are bad enough in application programs, such as word processors or spreadsheets, whose revised versions may no longer function the way their users expect them to; but changing the program that controls an artwork like *The Erl King* amounts to producing a new, different version of the work, which we were determined not to do.²⁴

²¹ Copying digital material technically creates successive generations as well, but one of the unique features of digital representation is that each copy can be identical to the previous one—and can easily be verified as such. This means that successive digital copies do not degrade, so the concept of generations is not applicable in the digital realm, as long as the encoding format of the copy is identical to that of the original. (If, however, the encoding format of a digital artifact is transformed in the process of making a copy, then this should be considered a new generation, which carries the attendant risk of degrading or corrupting the content of the original.)

²² Grahame Weinbren was a central participant in the decision to renew the work rather than create a new version of it, and he showed admirable restraint in resisting the temptation to improve or modify the work in the process of renewing it.

²³ Furthermore, the behavior of *The Erl King* is different each time it is used, since it depends on interaction with a human user. This would make it quite difficult to verify that a new program was indeed behaving the same way as the original in all cases.

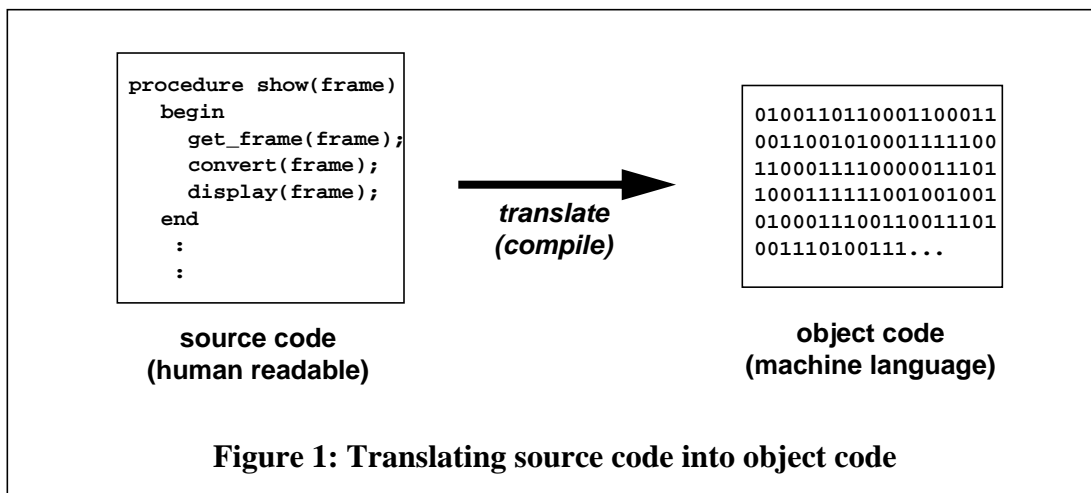
²⁴ For further discussion of the importance of running the original program that controls a digital work, see [2, 6, 7].

The Erl King's program consists of the Limosine-R runtime environment and the control logic (generated by using Limosine-A) that determines the behavior of the work.²⁵ The central challenge of our project was therefore to find a way of running *The Erl King's* runtime environment and control logic on a modern computer, since the original Sony computer was already obsolete and could not continue to run much longer. In addition, the audio/video content that is controlled by this code would have to be renewed in some way. Unfortunately, although modern computers are far more capable than the old Sony, they no longer run the Sony's CP/M operating system or programs written in the Pascal MT+ programming language. So it would require some work to make the original program run on a modern machine.

4.1 Source code versus object code

Because the program for *The Erl King* was written in a compiled version of the Pascal language, the alternatives for running the original program on a modern computer were to run its original “source code” or to run its original “object code”. Since this distinction is fundamental to much of what follows, it is explained here in some detail.

A compiled language program is written by a programmer in what is called source code, which is relatively easy for a human to understand but is too complex for a computer to run directly. Therefore, as illustrated in Figure 1, this source code is typically translated (or “compiled”) by a program called a compiler into “object code” which is the “machine language” of the computer on which it is to run. The machine language of a computer is the native tongue of the processor in that computer: it consists of much simpler commands than those found in most source code.²⁶ Unfortunately, different computers generally have different machine languages, so one computer cannot usually run object code intended for a different computer.



²⁵ *The Erl King* is an example of a “data-driven” program. Running it consists of running the Limosine-R runtime system, whose behavior is driven by data that represents specific control logic; this data is generated in advance by using the authoring system, Limosine-A.

²⁶ Object code is often called “binary” code, but this is somewhat misleading, since everything that is stored on a modern computer is binary, including text and human-readable source code.

Source code cannot usually be run directly on a computer, so programs are generally run as object code. In fact, most commercial programs are made available only in object code form, since the source code is more valuable as intellectual property, due to the fact that it can be more easily understood, modified, disguised, plagiarized or resold in pirated form. However, both the original Pascal source code and the compiled object code of *The Erl King* were available to us. Even though the source code could not be run directly, its availability provided an additional option for our preservation effort, as discussed in Section 5.2.

4.2 Additional preservation aspects

Even if a computer can run machine language programs intended for a different computer, a significant number of the instructions in many programs consist of requests to the operating system of the computer on which the program will run. An operating system (OS) is a program that helps a computer run other programs and provides services to those programs, such as helping them read the keyboard, display information on the screen, read and write disk files, etc. Therefore, a given program will probably not run correctly on two computers that use different operating systems, even if their machine languages are identical.²⁷

In addition to *The Erl King's* program, its analog video content would have to be renewed as well, since the Sony laserdisc players had become increasingly difficult to maintain. The program in the original work controlled the three laserdisc players in response to user interaction by selecting specific locations (frame numbers) within the content of each of their three video streams and directing the audio or video content from a given player to the output screen, starting from the given frame. The original audio/video content of *The Erl King* could be renewed in either analog or digital form, but in any case, it would have to be addressable by frame number as in the original system, so that the outputs of the control program would be interpreted correctly. This would also require reproducing the behavior of the custom-built video switcher device, which provided an additional hardware interface between the control program and the laserdisc players.

The Erl King's program also accesses text and image files that it uses to display overlays on top of its video content. These files are stored on the floppy disk drives of the original system, so their contents would have to be recreated and their formats understood in order to enable the renewed version of the work to access to them.

Finally, as noted above, *The Erl King* was developed using an authoring system (Limosine-A), which was a program that allowed the artists to develop the work, try it out, and modify it. Since this was an important and innovative aspect of *The Erl King*, the project team considered it an integral part of the work itself, even though the audience saw its effects only indirectly in the finished work. We felt strongly that the authoring system should be preserved in running form along with the work itself, to enable future art historians to see and try the tool that the artists had created to produce *The Erl King*. The authoring system—like the runtime system—was written in Pascal, and both its source and object code remained available, as they were for the runtime system.

²⁷ For example, both the Linux and the Windows(TM) operating systems can be run on many computers, but programs written for one of these operating systems cannot generally be run under the other, even on the same machine.

Therefore, whatever method was used to preserve the work itself (as embodied in the runtime system and its associated data files) should also be able to preserve the authoring system.

4.3 Summary of *The Erl King's* preservation characteristics

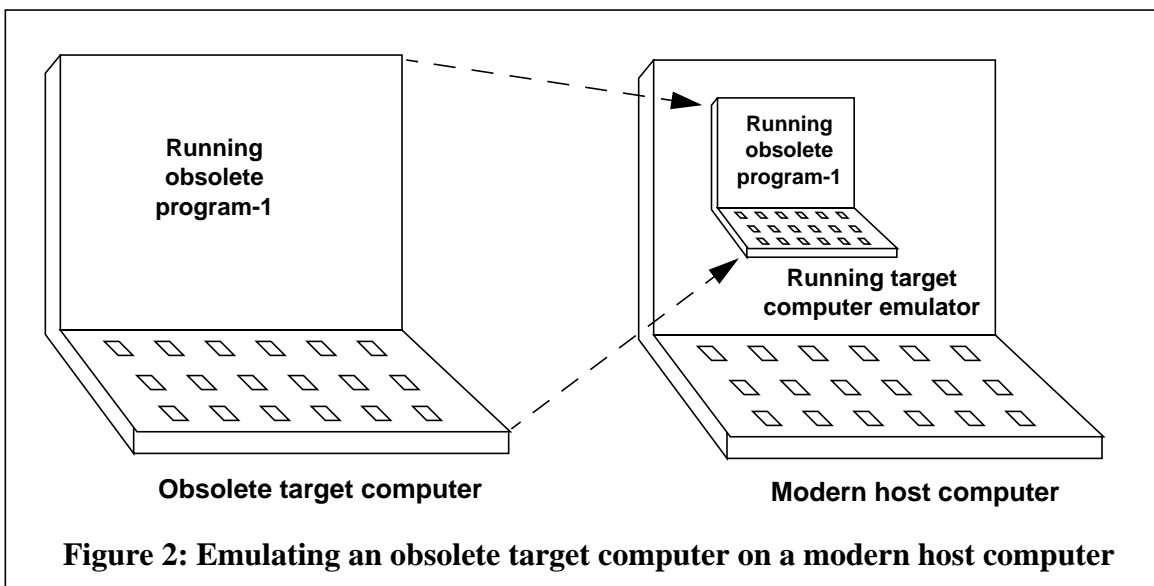
From a preservation perspective, *The Erl King* is a mixed analog/digital work that includes a computer, custom computer code, analog audio/video content derived from film, non-standard input/output (I/O) devices (i.e., the laserdisc players), and custom electronics (i.e., the video switcher). *The Erl King* is representative of a large class of analog/digital works of art, many of which similarly include custom software and hardware.

5. Alternative preservation strategies

Because all of the original hardware used in *The Erl King* had long since become obsolete, we decided not to try to use any of it in the renewed version of the work.²⁸ Since we had decided that it was crucial to run *The Erl King's* original software, we therefore had two alternatives.

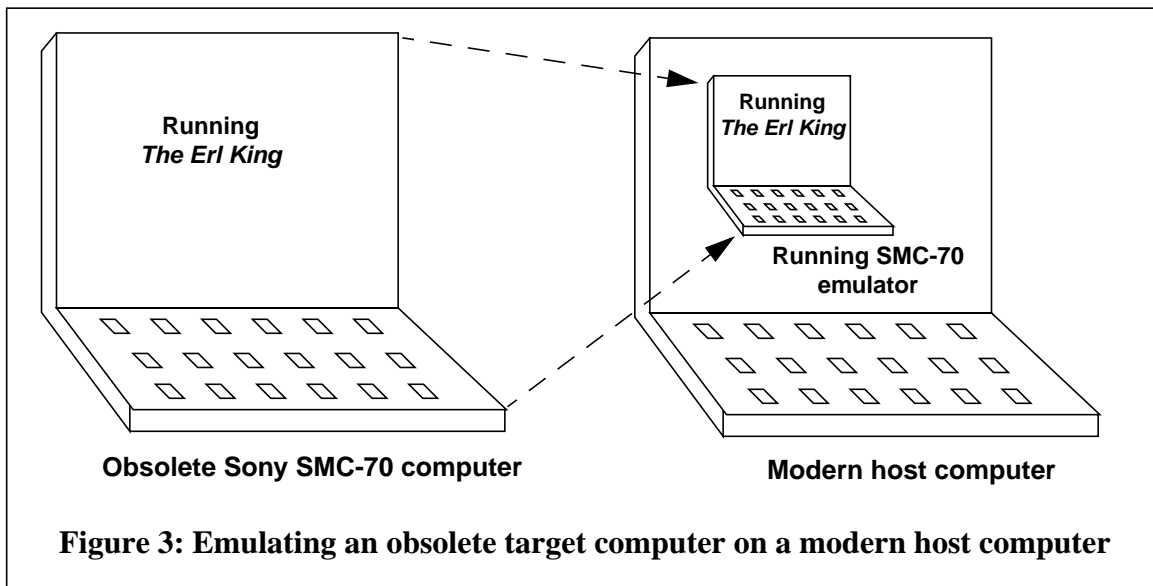
5.1 Emulator alternative

Our first alternative was to emulate the Sony SMC-70 computer on a modern computer in order to run the original object code for *The Erl King*. Hardware emulation of this kind is a time-honored and well-understood technique in which a program is written for one computer (referred to as the “host”) that enables it to run programs written in the machine language of a different computer (referred to as the “target”). If an emulator program of this kind is run on the host computer, it theoretically enables the host to run any program that was written for the target computer. This is illustrated in Figure 2. If we wrote a program that enabled a modern host computer to emulate the



²⁸ However, the Guggenheim now retains the original hardware as part of its permanent collection. For historical purposes, it may be important to preserve the original hardware of a computer-based artwork, even if it is no longer functional.

obsolete Sony target computer, it should allow us to run the original *Erl King* program on the modern computer, as illustrated in Figure 3. One major advantage of this approach is that it would also allow running the original CP/M operating system: since the object code of the OS itself is also in the machine language of the SMC-70, that code can be viewed as part of the full *Erl King* object program, which could be run under emulation on a modern computer. This is discussed further in Section 5.3 below.

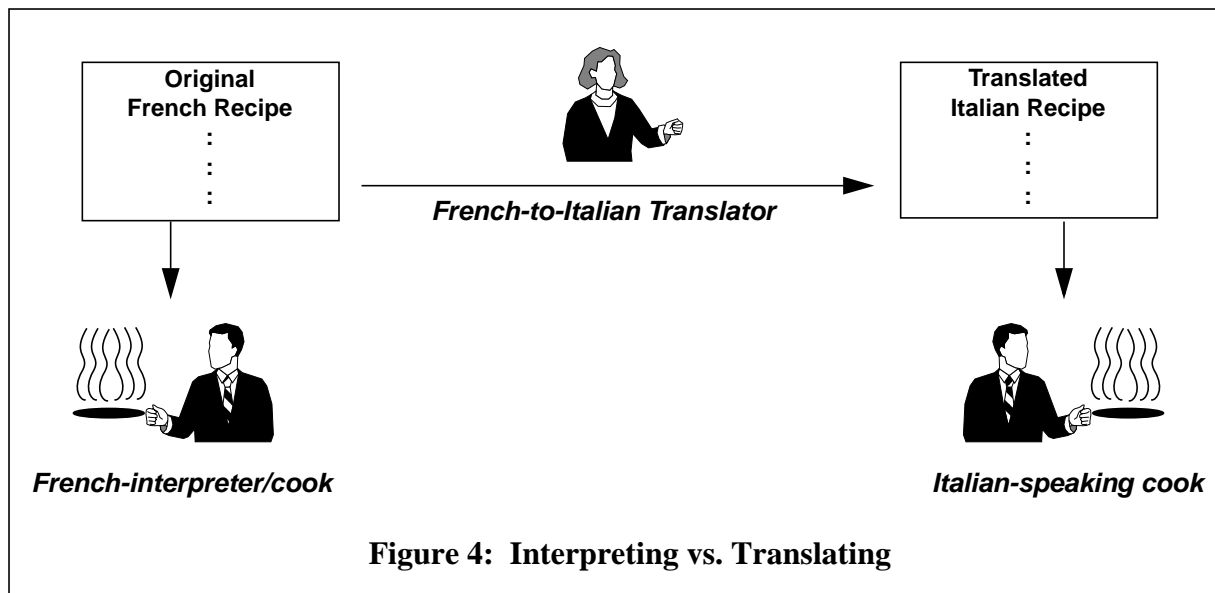


Actually, either hardware or software can be used to emulate either hardware or software, resulting in four cases: hardware emulating hardware, hardware emulating software, software emulating software, and software emulating hardware. An example of hardware emulating hardware is a modern “grandfather clock” that uses a quartz movement to emulate (and perhaps control) the pendulum of a traditional clock.²⁹ An example of hardware emulating software is a chip that performs the function of some program: we usually say that the chip “implements” the program in this case rather than emulates it, but the latter would also be correct. Any program that recreates the behavior of another program is an example of software emulating software: for example Sun’s StarOffice emulates the behavior of Microsoft Office, allowing Sun users to process files in Microsoft formats such as Word or PowerPoint without incurring Microsoft licensing fees. The fourth case, software emulating hardware, is the one that interests us here. Its advantage for preservation purposes is that software lasts longer than hardware, being immune to physical decay. Although the storage media on which software is stored do indeed decay, software itself, consisting of a string of bits, is as immortal as the written word, which endures despite the decay of any of its particular manifestations on paper, stone, etc. Unless otherwise stated, “emulation” will be used in the remainder of this report to mean the use of software to emulate hardware, since that is the case that is most relevant to preservation.

²⁹ Similarly, the Zilog Z80 processor can be thought of as a hardware emulation of the 8080 processor, since it executed the same instruction set as the Intel chip, albeit faster.

Computer scientists say that an emulator program “interprets” the machine language of its target computer. Interpretation here means simply that the emulator program examines each instruction in any target program and does whatever that instruction says to do. Since the distinction between an interpreter and a translator is important for the remainder of this discussion, the following analogy may be helpful.

The terms “interpreter” and “translator” are often used interchangeably in common parlance: for example, someone who can translate French into Italian as it is being spoken may be called an interpreter or a simultaneous translator. However, for this discussion, we define a “translator” as someone who can hear a sequence of instructions in one language (e.g., French), translate it into a different language (e.g., Italian), and speak the resulting translated instructions (i.e., in Italian). On the other hand, we define an “interpreter” as someone who hears that same sequence of instructions (e.g., in French) and simply carries them out, without speaking at all. Consider, for example, the instructions in a French recipe for a meal: a translator would convert the French recipe into Italian (thereby enabling an Italian speaking cook to prepare the meal), whereas an interpreter—whose native language need not be French—would simply follow the French instructions and prepare the meal directly. This is illustrated in Figure 4.



When we say that a computer “runs” or “executes” a program that is written in that computer’s machine language (i.e., in object code), we mean that the computer interprets the instructions in that machine language program. For example, a simple sequence of machine language instructions might tell a computer to add the numbers 1 and 2 and put the result in some location in the computer’s memory. When that sequence is executed, the computer interprets those instructions by adding 1+2 and putting the result in the desired location. Similarly, an emulator of this target computer would examine this same sequence of machine language instructions and would cause its host computer to add one and two and put the result in an appropriate location, even though the host computer could not interpret the original machine language instructions directly (i.e., without the emulator).

To reiterate, an emulator is a program that enables one computer to interpret the machine language of a different computer. Two computers that have different machine languages are like two people who speak different languages: for example, an English speaker might understand “one plus two” whereas a French speaker might understand “un plus deux”. An emulator program that enables an English speaker who does not understand French to interpret this simple French instruction would correspond to the English statement: “Interpret ‘un’ as the number 1; interpret ‘deux’ as the number 2; when you see ‘plus’ between two numbers, add them together”. An emulator program enables its host computer to “impersonate” some other, target computer. So long as the host computer is running the emulator program, it behaves just like the target computer, so it can run any program that the target computer could run.

Note that the BBC Domesday project, mentioned above, chose emulation as its solution, which turned out to be highly successful.³⁰

5.2 Source code interpreter alternative

The alternative to emulation was to write a program that could directly interpret the Pascal source code for *The Erl King*. Pascal MT+ was a compiled language; as explained above, a compiler is a program that translates source code into object code. This is analogous to the human translator above, who translates a French recipe into Italian to enable an Italian speaking cook to execute it. (In this analogy, the French recipe is *The Erl King’s* source code, whereas the translated, Italian recipe is *The Erl King’s* object code.) In contrast, a Pascal interpreter for a modern computer would be a program that simply did whatever the Pascal source code said to do; this is analogous to the human interpreter/cook above, who simply prepares the meal by following its French recipe.

Interpreting the source code for a compiled language is not necessarily any easier or harder than compiling that source code into object code. The original Pascal source code program for *The Erl King* was compiled into the machine language of the Z80 processor in the Sony SMC-70 by running the MT+ compiler program on the SMC-70. This produced object code for *The Erl King* in Z80 machine language. The resulting object code could then be run on the SMC-70 computer, producing the behavior of *The Erl King*. However, a modern interpreter for Pascal MT+ would have to be written decades after the Pascal MT+ compiler was written, so it would be difficult to verify that the interpreter was interpreting the meaning of the source code in exactly the same way that the original compiler translated it. In principle, one could run the original MT+ compiler to learn exactly how it behaved, but the Pascal MT+ compiler program itself, being a commercial product, was available only in object code. The version of this compiler that was used for *The Erl King* could therefore be run only on the computer for which it was intended, i.e., the Z80-based SMC-70. Running the MT+ compiler on a modern computer to explore its behavior would require an emulator for the obsolete Sony computer on which it originally ran, so this would require the emulator alternative again.³¹ Furthermore, because the source code of the MT+ compiler program is unavailable (as source code generally is for commercial programs), the exact behavior of the

³⁰. See [1] and [5].

³¹. Although the MT+ compiler can still be run on the original SMC-70 computer, that option will no longer be available once the SMC-70 becomes unmaintainable. In general, this option would not be available for old source code compilers, whose original computers are typically obsolete.

compiler cannot be gleaned from analyzing the source code that defines the compiler itself. This means that a Pascal MT+ interpreter must rely on the documentation for the Pascal MT+ language, which may or may not be correct.³²

The problem of verifying the behavior of an interpreter has its counterpart in the emulation approach, where the behavior of an emulator must be verified against that of the computer it emulates. Although it might at first seem that these problems are equivalent, it can be argued that the behavior of computer hardware is much more precisely and completely specified than the behavior of most programming languages and their compilers. This apparent contradiction arises from the fact that computer hardware is designed to be manufactured and then to be utilized by software designers. In order for something as complex as a computer's processor to be manufactured, its design must be specified in great detail: otherwise, it would be impossible to mass-produce working devices of this kind. In contrast, software—including compilers, which are simply programs that translate other programs from source code into object code—is not manufactured: instead, it is simply “written” and then copied, like a document. As a result, it is not strictly necessary to specify the design of a program in advance of writing it, and although it is considered good practice to do so, it is often done rather poorly. The correspondence between the specifications of a hardware device and the manufactured device itself is therefore typically much better than the correspondence between the specifications of a program and the program itself.

Similarly, descriptions of the behavior of computer hardware must be complete and accurate in order for software designers to write programs that run on that hardware and behave as desired: if the behavioral descriptions of hardware were not very precise and accurate, few programs could ever be made to work. In contrast, descriptions of the behavior of software are often quite poor: the software simply does whatever it does, regardless of whether it is described accurately. Evidence for this difference between hardware and software can be found in the fact that most computer hardware is spectacularly reliable (as it must be for anything to work at all), whereas most software is riddled with bugs and undocumented anomalies.

This asymmetry between hardware and software suggests that debugging and verifying an emulator should be at least as easy as debugging and verifying an interpreter. In either case, however, the process will be easier and more reliable if performed while the original hardware and software are still available and usable.

Pascal is a relatively simple language with reasonably well-defined semantics, so writing a valid interpreter for it is not as difficult as it might be for some other languages. But MT+ extended the standard Pascal language in a number of non-trivial ways, which increases the difficulty of interpreting MT+ source code. For example, MT+ allowed “inline” assembly code to be inserted in the middle of a Pascal program.³³ In our recipe analogy, inline code would correspond to an Italian phrase, such as “al dente” used in a French recipe; someone translating the French recipe into Italian would leave this phrase as is, since it is already in the target language; similarly, a French speaking interpreter would be expected to recognize the meaning of the Italian phrase and

³². It is not uncommon for programming language documentation to be incomplete or inaccurate in describing the details of how the source code for that language is translated into object code.

³³. Assembly code is a slightly more readable form of machine language.

perform its intended function. Inline code in an MT+ Pascal program was not translated or interpreted by the MT+ compiler but was simply turned into the equivalent machine language instructions, which were then executed directly by the computer when the compiled program ran.

Inline code of this sort was often needed in MT+ Pascal to communicate with I/O devices or ports, since there was no way to do this in the standard Pascal language. For example, *The Erl King* might have used such inline code to communicate with the Sony's audio cassette control ports, which were used to control the video switcher. In such cases, interpreting the Pascal source code would not be enough: an interpreter would also have to interpret any inline machine code that was present.³⁴ This would amount to emulating at least a subset of the Sony computer's Z80 machine language.³⁵

Furthermore, if the source code were interpreted, the behavior of any non-standard hardware devices, such as the touch screen and the laserdisc players, would still have to be emulated by software that performed equivalent functions, e.g., reporting to the program the screen location of a user's touch or displaying a video stream on the output screen, starting from a designated video frame. Similarly any custom-built devices such as the video switcher would have to be emulated by software. Emulating such non-standard or custom devices would be necessary in order to avoid changing the original *Erl King* program: the program would issue commands to—and receive input from—any such emulated devices just as it would have with its original hardware. That is, *The Erl King's* program would have to be fooled into thinking that its original hardware devices were still attached and were behaving as expected. Therefore even the source code interpreter alternative would require a certain amount of hardware emulation (i.e., the emulator alternative).

Finally, note that the source code interpreter alternative was viable for this project only because *The Erl King* makes relatively little use of functions provided by its OS. CP/M is a very simple OS compared to modern OSs, which provide extensive library and utility functions, windowing, and graphical user interfaces. If *The Erl King* had relied on CP/M for more complex functions, it would have been crucial to run significant parts of the OS itself in order to reproduce *The Erl King's* behavior. The interpretation approach would not have allowed doing this, since the source code for the OS was not available (as it rarely is). In fact, CP/M was written in assembly language (essentially machine language), so its source code is effectively identical to its object code; interpreting this code would therefore amount to emulating the Z-80 machine code, which again would involve the emulator alternative.

5.3 Additional Operating System and I/O device issues

It is typical to characterize a computer in terms of its operating system (for example, characterizing the Sony SMC-70 as a “CP/M computer”), but this is misleading. An operating system (OS) is

³⁴ As it turned out, *The Erl King* invoked the CP/M operating system to do most such operations, rather than using the inline feature of MT+ Pascal. However, as pointed out in Section 6.4, this does not relieve the interpreter of the need to interpret some machine language code.

³⁵ Interpreting machine language code is essentially what an emulator does, so an interpreter and an emulator are virtually the same thing when the language being interpreted is machine language. That is, emulating a computer consists of interpreting instructions in that computer's machine language.

merely a program that runs on the computer like any other program: its only distinction is that it provides services like those mentioned above, which are used by other programs, e.g., for accessing files and I/O devices. The simplest and safest way to build an emulator is to emulate the computer hardware itself and then run the object code of the original OS on the emulated machine: this is the most reliable way to ensure that the OS will behave as originally written. For example, the highly successful VirtualPC emulator that allows an Apple Macintosh computer to run the Windows OS simply emulates an Intel computer and then installs and runs an actual, licensed copy of Microsoft's Windows OS on the resulting emulated hardware.

In order to avoid licensing fees or legal issues, emulators sometimes supply their own version of an OS instead of running the original OS, but this is fraught with danger, since the surrogate OS program may not behave the same way as the original. Emulators of Z80 computers available on the web may recreate the CP/M OS in this way. However, a safer approach would be to emulate the Z80 processor and then run an actual copy of the original CP/M OS on the emulated machine. In principle, if one takes this approach it is not necessary to know anything about the operating system: the code of the original OS will run just as it did originally. In particular, there is nothing magical about the way an OS interacts with I/O devices, such as display screens or ports; however, in order to emulate an entire computer system, which includes such devices, it is necessary to emulate the ports and I/O devices themselves, so that they interact with the emulated computer just as the original devices interacted with the original computer.

The CP/M OS did not originally include any code for interacting with the disks or I/O devices of the computers on which it ran: such code (which came to be referred to as a Basic I/O System, or "BIOS") had to be added to the OS for each specific computer, since displays, ports, and other interfaces to I/O devices were not well standardized in the early days of the microcomputer era. Because any version of CP/M that was installed on a particular computer (such as the one in the Sony SMC-70) included such BIOS code, it would not in principle be necessary to understand this code or treat it in any special way in order to run the OS under emulation. Nevertheless, understanding the BIOS might be helpful in understanding the I/O device interfaces or port hardware in order to emulate them.

5.4 The approach chosen

The project initially decided to emulate the Sony SMC-70 hardware, since that seemed the safest solution to the problem and since even source code interpretation would require a certain amount of emulation, as explained above. Emulation would also be a more general solution, since not all similarly obsolete artworks are likely to have their source code available. Even if they did, their source code would be written in a wide range of languages other than Pascal (which was not a very popular language even in its heyday, when *The Erl King* was written). In order to use source code interpretation for all such works, interpreters would have to be written for each of their programming languages. In fact, even works written in other versions (or "dialects") of Pascal might require unique interpreters, since there was limited standardization across these dialects.

However, as the project proceeded, we concluded that we did not have sufficient resources and available emulation experience to undertake the emulator alternative.³⁶ We therefore decided instead to adopt the Pascal source code interpreter alternative to renew *The Erl King*. It is important to reiterate that this was viable only because *The Erl King*'s reliance on the CP/M operating system and system library or utility routines was fairly minimal.

As noted above, source code interpretation still required emulating any special I/O devices, as well as any inline machine code that appeared in the MT+ source code. The fact that neither of these efforts proved problematic argues that the emulator alternative might not have been problematic either. In fact, it is not clear whether emulating *The Erl King*'s hardware would have required more work than using its source code. In either case, it would have been necessary to emulate the behavior of *The Erl King*'s I/O devices—including its display screen, ports, LDPs, and custom video switcher—and to understand the input and output sequences that constituted the interfaces between these devices and *The Erl King*'s program. However, since we were unable to try both approaches, we cannot empirically compare the two alternatives.

In any case, as discussed in Section 7 below, the interpreter approach that we adopted for *The Erl King* was very successful. It should be stressed that this success depended strongly on the availability of excellent information about the original software and hardware and the active, supportive participation of Grahame Weinbren, who fortunately understood and remembered many of the details of the original implementation and had maintained excellent documentation. In addition, Isaac Dimitrovsky, the contract programmer for the project, did a superb job of designing and implementing the Pascal MT+ interpreter for *The Erl King*, as well as the many support programs that were required.

6. Implementation issues

Since the Pascal language was relatively well defined and because there was excellent documentation available for the MT+ variant of Pascal, writing an interpreter for MT+ was fairly straightforward. There was some concern that there might be undocumented subtleties of the MT+ compiler that would cause the behavior of the interpreter to differ from that of the compiled object code, thereby causing the interpreted version of *The Erl King* to behave differently from the original compiled version.³⁷ However, no significant cases of this sort have yet been discovered—though it is still conceivable that some may appear in the future as the interpreted version of *The Erl King* encounters some as-yet untested set of conditions and interactions.

Yet writing an interpreter was only part of the problem. All of the original hardware used in *The Erl King* would have to be either replaced by modern hardware or emulated by software. In

³⁶. An additional factor in our decision was the relative scarcity of computer-based artworks that rely on computers utilizing the Z80 processor. This scarcity reduces one of the potential benefits of the emulation approach, which is its ability to simultaneously preserve multiple computer-based works that depend on computers sharing the same processor. Emulation of a more modern processor (such as the ubiquitous Intel Pentium) would be more advantageous, since many more computer-based artworks have been designed for computers using that processor. Emulating the Pentium would therefore facilitate preserving a large number of artworks, whereas emulating the Z80 would not have done so.

³⁷. Some minor subtleties of this sort were discovered and addressed, as discussed in [3].

addition, we wanted to convert the original video content of *The Erl King* into a more modern form that could be used by current and future generations of computers. Finally, as discussed above, two of the trickiest aspects of the interpreter approach to renewing *The Erl King* involved the interrelated problems of interpreting any inline Z80 machine language (actually assembly language) code in the Pascal source program and emulating the behavior of custom hardware devices, such as the video switcher. The following subsections discuss these implementation issues in further detail.

6.1 Replacing obsolete hardware

As explained above, one of the basic premises of our approach to renewing *The Erl King* was that the original hardware that was used to implement it was not essential to its artistic effect. Because hardware is difficult to preserve in working order, its inessential aspect was fortunate, since it gave us the flexibility to use new computers, new displays, and new video-delivery hardware wherever necessary, so long as these reproduced the behavior of the original work. Since we were also committed to running the original *Erl King* program to control the behavior of the work, we felt that the result would be very faithful to the original. The program would behave just as it always had, and although it would be controlling new hardware, the resulting interactive experience for the user should be virtually identical to that of the original work.

Whether we emulated the Z80 processor to run *The Erl King's* object code or interpreted its source code, computing speed would certainly be important. Both emulation and interpretation can easily be ten times slower than running an equivalent compiled object code program. Fortunately, however, modern computers are so much faster than the Z80 processor in the SMC-70, that a mere factor of 10 (or even 100) would not present any problems. In fact, most modern personal computers are thousands of times as fast as the Z80. In practice, we would be more likely to have to slow a modern computer down artificially to match the speed of the original SMC-70, as indeed turned out to be the case.

In order to achieve the desired level of fidelity, certain key aspects of the original hardware would have to be preserved, whether they were implemented in new hardware or in software. For example, the program would have to run at the same speed as the original, and the displayed video would have to appear identical to the original, in terms of its smoothness, color, resolution, etc. Similarly, all delays and latency would have to appear identical, including the time required to switch among video streams when the user touches the screen.

The Erl King's original Carroll touchscreen would have to be replaced by a new, equivalent device. The original touchscreen detected the presence of one of the user's fingers by means of an array of light-detecting sensors in a frame around the screen, rather than sensing the pressure of a touch on a transparent panel overlaying the screen, as most modern touchscreens do. In order to retain as much of the original feel of this interface as possible, it was decided to use a modern infrared touchscreen rather than one of the more common pressure-sensitive panels. Since the resolution of such modern touchscreens is higher than that of the original, it was necessary to translate the finer-grained screen coordinates returned by the modern screen into the relatively coarse coordinate system of the original screen. In addition, the original *Erl King* program used a "polling loop" to continually interrogate the touchscreen to see if a touch was in progress; in contrast, the

driver programs for modern touchscreens generate signal “events” whenever the user touches the screen. It was therefore also necessary to convert these instantaneous events into longer-lived state changes that could be detected by *The Erl King’s* polling mechanism.

6.2 Converting the video content of *The Erl King*

The original laserdisc players and the laserdiscs themselves that were used in *The Erl King* are now obsolete, despite the fact that they can still be coaxed into working. We therefore felt that the video content of *The Erl King* should be converted into some more modern form in order to preserve it, even though this would introduce an additional copy generation. Whether we converted the video into a modern form that was analog or digital, it would inevitably involve some loss of quality. Digitizing analog material involves “sampling” the analog information; the higher the sampling rate, the better the resulting quality, but some loss is difficult to avoid, and various digitization “artifacts” may be introduced in the process.³⁸ The technical issues involved in converting *The Erl King’s* video into digital form are presented elsewhere,³⁹ but they are reviewed here.

Since the original audio/visual material for *The Erl King* had been shot on 16mm color sound film, the artists decided that there was nothing sacred about maintaining the *The Erl King’s* analog video format. In fact, because the 3:2 pulldown process that had been used to convert the original 24 fps film to 30 fps video had arguably already distorted the look of the original material somewhat, we considered the possibility of going back to the original 16mm film source instead of using the analog video from the laserdiscs. However, the 16mm film was no longer in suitable condition to be used, so this alternative was discarded almost immediately. Nevertheless, we felt it would be advantageous to convert back to 24 fps if we could do this without additional loss.

The 1-inch magnetic videotape masters containing the synchronized audio and visual streams of *The Erl King* (described in Section 3 above) were still available, so we used these as our source in preference to the laserdiscs themselves, which would have been lower quality. We dubbed the contents of the 1-inch masters onto Digibeta format tape and used the Digibeta cassettes as the starting point for subsequent audio and video processing.⁴⁰

One additional complication was introduced by reversing the 3:2 pulldown process. This involved the fact that the frame numbers used by *The Erl King* to address specific sequences in the video material had to be translated to allow addressing the proper sequences in their 24 fps form. This is discussed in further detail in [3].

³⁸ For example, digitized images may appear to be “pixelated” or may display jagged “sawtooth” patterns in place of smooth edges for lines that are angled between horizontal and vertical (for unobvious reasons, this is called “aliasing”).

³⁹ See [2].

⁴⁰ Sony’s Digital Betacam (Digibeta) is a high-quality, professional digital format recorded on 1/2-inch tape cassettes; it was designed for high-end applications in standard definition television.

Early in the discussion of the video playback issue, we considered simply extracting an individual image file for each frame, and then displaying these as sequential images at the desired frame rate. This would be much easier than using a modern compression scheme (such as MPEG), since it would avoid the problem of knowing which frames were “key frames” that might be addressed by *The Erl King’s* program when it ran (the issue of key frames is elaborated below). It would also provide a simple way of returning to a 24 fps frame rate. The computer and disk speed requirements for displaying individual frames as separate images would be considerable, but it seemed worth at least trying this option to see if a modern PC could accomplish it, before resorting to more complex solutions. This straightforward approach was initially shelved but was later re-examined and ultimately adopted.

On the way to this straightforward solution, however, we considered a number of other options. One of the first to be rejected was simply to transfer the analog video to digital DVD. Several artists and video experts whom we consulted agreed that the DVD medium does not have the necessary access speed to allow it to be used as a replacement for the original laserdiscs. Nevertheless, most project members felt that it would be best to convert the analog video into digital form, since that would arrest its further decay and would make it more likely to work under future preservation schemes.

Video servers were also considered, but their interfaces were felt to be both too complex and too constraining for our needs. In addition, the use of modern video compression schemes, such as MPEG, introduced the problem of how to access specific frames. *The Erl King* addresses its video content by frame number, whereas MPEG normally allows quick access only to “key” frames, which are stored in full detail. For compression purposes, a key frame is normally one whose visual content changes drastically from its predecessor; all frames in between key frames can then be compressed very effectively by representing only the changes (“deltas”) between each frame, which are relatively minor. However, randomly accessing a frame that is in between key frames requires extra work: the system must back up to find the last key frame prior to the given frame and then apply all of the successive changes for all intermediate frames in order to render the desired frame. Although it is possible to force MPEG to treat every frame as a key frame, that would undermine most of its compression power and would amount to storing each frame as a separate image file—whereas the MPEG format is not ideal for that purpose.

Ultimately, we determined that saving each frame as a separate image file was a sufficient solution; however, we stored these images as uncompressed 640 by 480 pixel Microsoft Windows BMP (bitmap) files rather than MPEG, which allowed them to be displayed faster.⁴¹ Using this approach, displaying a video segment starting from a given frame number simply requires displaying the starting image file first and then displaying each successive image file in sequence, at the desired frame rate. Modern desktop computers and hard disk drives are fast enough to perform this access and display at full video rates without undue stress. Because this approach gave us the flexibility to use our desired frame rate, we also converted back to 24 fps, thereby producing what is

⁴¹ Displaying a compressed image requires considerable computation to decompress the image prior to rendering it on the screen. It is therefore possible that using any compressed image format would have increased the computational burden on our frame-by-frame display scheme to the point of making it untenable.

presumably smoother video output than the original work, since we eliminated the artifacts of the 3:2 pulldown conversion.⁴²

Converting the analog video did require some additional effort, in order to translate the frame numbering of the 30 fps stream into the desired 24 fps stream. However, since the 3:2 pulldown process does not eliminate any of the original 24 fps film frames, we were able to perform this reverse conversion without losing any of the correspondence between the video content and the frame number access logic in *The Erl King's* program.

The Erl King addresses specific frame numbers in each of the three LDP streams, using the video switcher to select which LDP sends its output to the display screen. In order to eliminate the video switcher and the LDPs without disturbing the behavior of the program, we therefore had to create simple software emulations of these devices, to direct the selected stream of frames to the display as desired. Doing this involved creating three image frame streams to mimic the three LDPs and interpreting *The Erl King's* output appropriately to ensure that the desired images were displayed. In principle, all of the video content could be stored on a single hard drive, but it turned out to be necessary to put the frame sequences for each video stream on a separate drive to minimize delays in switching among these streams (just as the three LDPs had originally been used to avoid such delays). Using this scheme, a set of image files on one of our three hard drives emulates the video stream on one of the original LDPs.⁴³

6.3 Digitizing the audio content of *The Erl King*

Each of *The Erl King's* three original laserdiscs contains two audio tracks. Converting this audio content into digital form was straightforward. Each of the six audio tracks was digitized as a single file of uncompressed 16-bit stereo, at a sampling rate of 48 Kbps (thousand bits per second). This is a common rate for high fidelity sound, which resulted in negligible loss from the original analog audio on the laserdiscs.

The original *Erl King* accesses its audio tracks the same way it accesses the video content of the laserdiscs: by addressing a frame number on the disc, selecting the desired track, and issuing a command to play the selected audio track from the selected frame on the selected laserdisc. In order to address desired audio segments in the digital sound files of our renewed version of *The Erl King*, we translate the original laserdisc frame numbers into the equivalent time offset from the beginning of the desired track, and the interpreter begins playing the sound file for that track from that offset.⁴⁴ For example, if *The Erl King's* program issues a command to play sounds starting at frame number 3060 on track 2 of laserdisc 3, and if frame 3060 corresponds (at 30 fps) to 102 seconds

⁴² It might be argued that this choice violates our intent to preserve the original *Erl King* as exactly as possible; but we felt that this approach instead restored the work to its intended form, since the original choice to convert the 16mm source film to 30 fps analog video (which required the 3:2 pulldown conversion) was simply an expediency.

⁴³ Note: a "hard drive" is simply a magnetic disk. Prior to the invention of the floppy disk, all magnetic disks were hard (i.e., made out of rigid material), though no one thought to refer to them that way.

⁴⁴ The Sound API (Application Program Interface) of the Java programming environment in which our interpreter was implemented provides a simple way of playing sound from a file, beginning at any point in that file.

from the beginning of the disc, the interpreter starts playing the sound file for track 2 of laserdisc 3 beginning 102 seconds from the start of that file.

6.4 Interpreting machine code and emulating custom hardware

Most computer programs—and certainly most computer-based artworks—need a way to control external I/O devices. For example, in order to switch video streams, *The Erl King's* program sends short commands (sequences of bits) to the SMC-70's two audio cassette ports, whose outputs are attached to *The Erl King's* custom-built video switcher. As discussed in Section 5.2 above, the standard Pascal language provided no way of addressing such ports, but the inline feature of Pascal MT+ could be used to embed machine language instructions in the Pascal source program; the MT+ compiler recognized these as machine-language instructions and arranged to pass them to the Z80 processor for execution at the appropriate time when the object program ran. When the Z80 executed those instructions, they would cause the appropriate bits to be sent to the audio cassette ports, thereby controlling the video switcher. As it turned out, *The Erl King* made little or no use of this inline feature: instead, it invoked special system programs provided by the CP/M operating system to interact with the SMC-70's ports. However, neither MT+ Pascal nor CP/M contained standardized system programs of this sort: they were a special feature of the Sony SMC-70 version of CP/M. Invoking such system programs would therefore not have been an option for most other programs written in MT+ Pascal or running under CP/M on other computers.

Furthermore, the availability of these special system programs on the SMC-70 computer does not in itself enable an interpreter to recreate *The Erl King's* interactions with ports or “peripheral” (external) devices. The problem is that the supplied system programs are most unlikely to be written in the same programming language as the main program—which is the language that the interpreter is designed to interpret. In fact, such system programs are likely to be written in the machine language of the computer, not in a higher-level language such as Pascal. Moreover, even if they were written in Pascal, their source code would be proprietary (to Sony in our case) and so would not be available to the interpreter. Therefore, in order for a source code interpreter to reproduce the behavior of such system programs, it must understand the machine code in which they are written—which amounts to the same thing as interpreting inline code embedded in the Pascal source program. Either way, an interpreter must handle machine code in order to interact correctly with ports and peripheral devices.

There are at least two ways of handling such machine code in a source code interpreter. A strict emulation approach would emulate the Z80 machine language instructions when they were encountered; but this would also require emulating the behavior of the audio cassette port hardware and the behavior of the video switcher. Although none of this would be very difficult, it would be something of a waste, since the sole purpose of all of this in the original program was to select among one of the three LDP video streams, whereas the renewed version of *The Erl King* would not utilize LDPs at all. Alternatively, whatever mechanism replaced the LDPs in the renewed version could be controlled directly by the interpreter at the appropriate places in *The Erl King's* program, without bothering to use Z80 instructions or to pretend to communicate with what would now be the non-existent audio cassette ports, video switcher, or LDPs.

This issue is quite important, since it generalizes to the preservation of any program that controls external (peripheral) devices that were attached to the original computer on which that program ran. In many cases, those peripheral devices will no longer exist in a modern recreation of the original system: instead, their roles will be performed by modern I/O or storage devices (i.e., modern hardware emulating obsolete hardware) or by additional programs (i.e., software emulating obsolete hardware). For example, the LDPs in the original *Erl King* are replaced in the renewed version by magnetic disks containing digitized images, whereas the video switcher is replaced by a simple program that selects which video stream to direct to the output display. When modern hardware is used to emulate obsolete hardware, the interfaces that control the modern hardware are likely to be different from those of the original hardware that it replaces. Yet if the original program is not to be changed, the original interface of that program to its original hardware must not be changed: the original program must issue its original commands to the emulated devices and get its original inputs from them in return. For example, *The Erl King's* program must get its expected input whenever the user touches the screen, even if a modern touchscreen (or some other pointing device, such as a mouse) replaces the original touchscreen. Similarly, *The Erl King's* program must be able to issue its original output commands to control the audio and video streams, and these outputs must produce the same effects that they produced in the original system.

An efficient solution to this general problem is to interpret the inputs and outputs of the original program to cause the peripheral devices of the modern system to behave as necessary so as to reproduce the desired effects of the original system. For example, rather than emulating an inline sequence of Z80 machine language instructions to send specific bits to emulated audio cassette ports so as to control an emulated video switcher device that controls emulated LDPs which display a desired video segment, the original sequence of inline instructions can be interpreted as a direct command to the modern image-display subsystem to display the desired video segment. This process must reproduce any delays or other relevant aspects of the original interface, but so long as the original program is successfully fooled into thinking it is still connected to its original devices, the system should behave as intended. As described in further detail in [3], this approach was adopted for the *Erl King* renewal effort and was found to work quite well.

6.5 How close is close enough?

Any conservation effort must face the issue of how faithful to the original artwork its results can and should be. Modern conservationists try to use restoration techniques that can be easily detected and reversed, to prevent permanently corrupting an artwork and to allow the use of improved techniques in the future. Our renewal effort used a modern computer, modern peripheral devices, and modern audio/video formats in place of the originals, though it retained the original code of the *The Erl King*. The result is not identical to the original, because the exact characteristics of the display screen, sound system, frame rate, etc. are different. However, it is physically impossible to keep old computer hardware running forever, and in any case its characteristics change over time. For example, *The Erl King's* original touchscreen no longer responds as reliably as it once did, and the color, brightness, and contrast of the work's original display have undoubtedly changed as it has aged. So it is impossible to ensure that any computer-based artwork will continue to behave identically forever. Renewal techniques like those discussed in this report can approximate the behavior of an original artwork, but they can never duplicate them exactly. Any such renewal or restoration effort must therefore accept a certain degree of imperfection, and artists and curators

must ultimately decide how much imperfection can be tolerated in each case. The Variable Media Network attempts to address such questions, not only for digital artworks, but for ephemeral and “variable” works of all kinds.

In considering these issues for computer-based artworks, it would be useful to identify relevant dimensions of behavior, each of which may define acceptable or unacceptable levels of fidelity. For example, such dimensions might include the timing of an artwork’s interactions, the fidelity of its color reproduction, the frame rate, refresh rate, size, contrast, and resolution of its display, the frequency range, dynamic range, and other characteristics of its sound, and any relevant haptic or tactile aspects of its user interface. Additional dimensions (such as 3-D or olfactory) can be added in the future as additional sensory modes are utilized. Appropriate measures of fidelity could be devised for each such dimension, and target levels of these measures could be defined for any given conservation effort. In the absence of such objective criteria, the question “How close is close enough?” must be decided subjectively by artists, curators, and audiences.

6.6 The bottom line

Conservation and restoration efforts are often undertaken and justified on a case-by-case basis. Our *Erl King* renewal effort was similarly performed as a unique task. Creating the Pascal interpreter and associated software required several hundred hours of expert programming; in addition, the Guggenheim contributed in-kind support, and individual members of the project team donated additional effort. If future, similar renewal efforts benefit from our experience, they may require less investment. This would certainly be true for any digital artworks based on programs written in Pascal, since our interpreter would facilitate renewing these at considerably lower cost; however, there are not likely to be many such artworks. If we had used emulation, the resulting emulator would have facilitated the renewal of other digital artworks utilizing the Sony SMC-70 computer or other computers using the Intel 8080 or Zilog Z80 processors. There may be somewhat more such artworks than those whose programs are written in Pascal, but this is still likely to be a small set. Future emulation of more recent processors (such as the Intel Pentium or the PowerPC used in recent Apple Macintosh computers) might have greater leverage, since a much larger number of digital artworks run on computers that utilize these modern processors. Nevertheless, the cost of renewing or restoring a digital artwork may be significant—especially if the renewal effort requires converting analog content into digital form or recreating customized hardware—even if high-leverage techniques like emulation are used. Ultimately, conservationists and artists must decide whether such techniques are worthwhile and which artworks are worth renewing or restoring by these means.

7. Results

The renewed version of *The Erl King* was displayed alongside a reinstallation of the original work at the Guggenheim's Seeing Double exhibit, as part of the Variable Media Network effort.⁴⁵ This show was analyzed and discussed in the symposium *Echoes of Art: Emulation As a Preservation Strategy*, held at the Guggenheim on May 8, 2004.⁴⁶ The exhibit included a number of other old computer-based artworks alongside renewed versions of themselves. That is, each work in the show was present in both an original and a renewed form.⁴⁷ Since the *Erl King* renewal effort was the most ambitious of those exhibited, it was the centerpiece of the show. Although the stated theme of the show was emulation, *The Erl King* was renewed by the related—but distinct—approach of source code interpretation. However, this can be justified by arguing that the underlying premise of the show was that the best way to preserve a work of computer-based art is to run its original program if at all possible, whether that is achieved by running the object code of that program under emulation or by interpreting its source code.

For the purposes of the exhibit, the dual, side-by-side implementations of *The Erl King* were displayed with their hardware visible. This violated the artists' original intent somewhat, since the implementation was not considered part of the essence of the work. However, for the purposes of the exhibit, it was useful to make the hardware visible, so that the audience could see the difference between the two implementations. In fact, the results of the renewal effort were so good that the two implementations were virtually indistinguishable except for the look of their hardware.

It is difficult to imagine what someone who wandered into the exhibit off the Guggenheim's main spiral would have made of the two versions of *The Erl King*. Without reading the explanatory signage, a casual observer might well have taken the "Seeing Double" title of the show literally and wondered why we had bothered to display two instances of the same work next to each other.⁴⁸ On the other hand, those of us who were involved in the project were understandably thrilled to see that the behavior of the renewed version of *The Erl King* was virtually identical to that of the original.⁴⁹

⁴⁵ See <http://www.variablemedia.net/e/seeingdouble> or <http://www.guggenheim.org/exhibitions/emulation/index.html>, as well as [4].

⁴⁶ A transcript of this symposium is available at <http://www.variablemedia.net/e/echoes/index.html>.

⁴⁷ Some of the works in the show were renewed by means of migration rather than emulation; that is, their code was rewritten for a modern computer. The curators included these works to suggest that emulation might not be the best preservation strategy for every computer-based artwork.

⁴⁸ A survey given to 40 visitors to the exhibition concluded that the renewed version of *The Erl King* did a good job of conveying the experience of the original.

⁴⁹ As pointed out above, it is generally impossible for any renewal or restoration effort to produce a result that is absolutely identical to an original artwork. Furthermore, since *The Erl King* is interactive, the side-by-side versions rarely showed the same content at the same moment, unless they were restarted simultaneously and allowed to run without human interaction or two users synchronized their interactions with the two versions to produce identical sequences of behavior. Nevertheless, it is not an overstatement to say that, for all practical purposes, the two versions of *The Erl King* behaved identically.

8. Conclusions

The Erl King is representative of a growing class of artworks that employ computers to interact with users to control multi-media presentations. Many such works that were developed in the 1980s or even 1990s made use of mixed analog/digital content, before the widespread availability of fully digital sound and video. Some such works also utilized custom-built hardware devices to perform digital or analog functions like those of *The Erl King's* video switcher. Newer works are more likely to be fully digital, since the functions of custom analog hardware devices and the behavior of analog media can in most cases now be performed more easily (and affordably) by programs running on off-the-shelf digital computers attached to off-the-shelf input/output devices.

The preservation strategy that we adopted for *The Erl King* can be applied to most digital or mixed analog/digital artworks. In each case, it must first be decided whether any original analog content should be converted to digital form. Doing so will generally make it easier to manipulate that content, as well as simplifying its possible conversion into future digital formats, as may be needed for future conservation efforts. However, in some cases, converting analog content to digital form may lose or corrupt essential aspects of the original content; and even if nothing essential is lost, digitizing analog material may result in a reduction in quality, the introduction of digitization artifacts, or subtle differences in the look-and-feel of the material.

Similarly, it must be decided whether any custom-built analog hardware should be emulated by software. (This question applies mainly to analog hardware, since any original digital hardware, whether custom-built or not, should lose nothing by being emulated by software, assuming that the emulation can reproduce the original hardware's function and speed.) Emulating analog hardware in this way amounts to converting it into digital form. As with analog content, the advantage of converting analog hardware to digital form by emulating it in software is that it becomes easier to manipulate and to convert into future digital formats, as may be needed for future conservation efforts. However, it is possible that emulating analog hardware in software may introduce subtle differences in behavior, resulting from digitization effects.

For fully digital or analog/digital artworks of these kinds, whose behavior is controlled by a computer program, the safest preservation strategy seems to be to run the work's original program, to ensure that the logic of its behavior is not changed. Care must be taken, however, to ensure that the original program's interfaces to its original input/output devices are interpreted correctly, in order to produce equivalent results on modern I/O devices. In many cases, it should be possible to emulate original I/O devices by reproducing their logical behavior without needing to emulate (or even understand) their internal workings. The code that controlled the original work can then be run without modification, so long as it is interpreted correctly by the emulated versions of the I/O devices to which it is connected.

As discussed in this report, there are essentially two ways to run the original code of an artwork's control program on a modern computer. The more generic approach is to emulate the artwork's original computer, so that the artwork's original object code can be run under emulation on the modern computer. The alternative, as used for *The Erl King*, is to write an interpreter for the source code of the artwork's original program and then run the original source code under this interpreter on the modern computer.

The interpreter approach has some potential disadvantages: (1) source code for an artwork's original program may not be available; (2) the programming language in which the artwork's original source code was written may be poorly documented, and the actual behavior of programs written in that language may not have conformed to its specifications; (3) a different interpreter must be written for each programming language in which any artwork's original source code was written; (4) some emulation is likely to be needed even if the interpreter approach is taken, in order to deal with low-level interfaces to input/output devices; and (5) the interpreter approach cannot reproduce the behavior of the original operating system under which the artwork originally ran, since the source code of that OS is most unlikely to be available.⁵⁰

Since the interpreter approach was the one that was adopted for *The Erl King*, and since it worked so well, it is instructive to ask why these potential disadvantages were not problematic in our case. The answers are: (1) source code was available; (2) the programming language was unusually well documented and well behaved; (3) only a single artwork was being addressed, so only a single interpreter had to be implemented; (4) the small amount of emulation required to deal with input/output devices did not prove to be a burden; and (5) it was not necessary to reproduce most of the behavior of the CP/M OS, since *The Erl King* did not rely heavily on functions provided by CP/M.

Nevertheless, future preservation efforts of this sort are likely to find emulation a more general and safer solution, especially if source code for the artwork in question is unavailable or if the artwork relies to a significant extent on functions provided by the OS under which it originally ran.

Our *Erl King* renewal effort has broad implications for the preservation of mixed analog/digital or purely digital works of art. In particular, it shows the value of retaining the ability to run the original program that controls or generates such a work. In the future, computer-based artworks are likely to rely more and more heavily on sophisticated programming, making it all the more important to retain the precise behavior of their original programs. The only way to guarantee that this behavior is retained is to run those original programs, without rewriting them or converting them in any way. Emulation—and to some extent, source code interpretation—can retain the ability to run original programs in just this way.

The most difficult aspect of preserving or restoring computer-based artworks involves their use of special purpose I/O devices or other peripherals. In order for any program to produce an effect or result that can be perceived by a human observer, it must ultimately deliver output of some kind; similarly, in order for a program to be controlled by or interact with a human, it must accept input of some kind. Any computer-based artwork must therefore make use of at least one output device, and if the artwork is interactive, it must also make use of at least one input device. Additional peripherals (such as storage media like laserdisc players or DVD drives, network connections, etc.) may also be required. Many artworks use specialized or custom-built devices to achieve their desired behavior; and even if an artwork employs only standard devices, these may become non-standard in the future, as *The Erl King's* laserdisc players did. As I/O and other peripheral

⁵⁰ Even if source code for the OS were available (as, for example, it might be for Linux), the OS would probably not be written in the same programming language as the artwork itself and would therefore require writing a separate interpreter merely to run the OS.

devices become increasingly digital, emulating their behavior will become easier; yet every I/O device must ultimately interact with the external world, which is analog (for example, we perceive sound and light waves as analog phenomena). So the preservation problem will always include both analog and digital components.⁵¹

I/O and other peripheral devices constitute additional hardware that must be preserved, in addition to the processor that is the heart of any computer system. The program that controls a computer-based artwork runs on the processor in its computer, but this program must communicate with I/O and other peripheral devices by sending and receiving command sequences that are understood by the devices.⁵² Emulating a computer-based artwork will therefore generally require emulating all of its I/O and other peripheral devices as well as the processor in its computer. If modern devices are substituted for an artwork's original devices, they are unlikely to use the same command sequences, so these sequences must be understood and interpreted or translated appropriately in order to recreate the behavior of the artwork.

Because of their potential reliance on I/O and other peripheral devices, each computer-based artwork must be considered a special case from a preservation perspective.⁵³ Each preservation or restoration effort for a computer-based artwork should therefore be approached with an open mind, since no single technique may be appropriate for all such works. Nevertheless, techniques that allow running an artwork's original program—namely, emulation and to some extent source code interpretation—offer unique assurance, if not an outright guarantee, that the artwork will retain its original behavior.

⁵¹ However, it may not be necessary for preservationists to worry about analog aspects of a work in all cases. For example, if an artwork produced digital sound outputs that were intended to be played on standard external devices that were not considered part of the work itself (e.g., MIDI sound systems), then it might be possible to ignore the conversion of the artwork's digital outputs to analog sound when preserving the work. This is analogous to saying that a recorded piece of music need not include the playback system or speakers that will be used to render it as sound.

⁵² Some peripheral devices may contain their own processors and run their own programs, and these programs may even be sent to a peripheral device by the computer itself. However, such cases can be handled by emulation, simply by emulating the processor in the peripheral device as well as the processor in the computer itself.

⁵³ This is no different from traditional preservation, in which each individual artwork is considered a special case, at least to some extent.

References

1. Darlington, Jeffrey, Andy Finney and Adrian Pearce, "Domesday Redux: The rescue of the BBC Domesday Project videodiscs", 30-July-2003, *Ariadne*, Issue 36, <http://www.ariadne.ac.uk/issue36/tna/intro.html>.
2. Depocas, Alain, Jon Ippolito, Caitlin Jones, ed., *Permanence Through Change: The Variable Media Approach*, ISBN: 0-9684693-2-9, The Solomon R. Guggenheim Foundation, NY and The Daniel Langlois Foundation for Art, Science, and Technology, Montreal, 2003, www.variablemedia.net/pdf/Permanence.pdf.
3. Dimitrovsky, Isaac, *Final report, Erl-King project*, <http://www.variablemedia.net/e/seeingdouble/report.html>, April 1, 2004
4. Jones, Caitlin, *Seeing Double: Emulation In Theory And Practice — The Erl King Case Study*, Presented at the Electronic Media Group, Annual Meeting of the American Institute for Conservation of Historic and Artistic Works, Portland, Oregon, June 14, 2004.
5. Mellor, Phil, *CAMiLEON: Emulation and BBC Domesday*, RLG DigiNews, April 15, 2003, Volume 7, Number 2, <http://www.rlg.org/legacy/preserv/diginews/diginews7-2.html#feature3>, ISSN 1093-5371.
6. Rothenberg, Jeff, "Ensuring the Longevity of Digital Documents," *Scientific American*, Vol. 272, Number 1, pp. 42-7, January 1995.
7. Rothenberg, Jeff, *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*, ISBN 1-887334-63-7, January 1998.