文学的テキストとその視覚的表現の可能性について

Visualizing Literary Text

Katsuaki TAIRA

What is the relationship between what we perceive and the emotional and intellectual content it evokes? That may be a rather sweeping question because is there really universally agreed evocative mechanism in which any given "trigger" automatically elicits homogeneous reactions in any randomly selected groups of people? Unless we go truly philosophical and abstract the answer, from an empirical standpoint, may be a resounding no. However, before we drop such an important question I posited in the very first sentence of this essay, let us pause a minute and consider the implications of such mechanism if it could have even a modicum of a chance to establish in this living world, and hopefully apply it to evoke the maximum strands of those subtle and nuanced evocations poets and other creative writers try hard to realize in their ever elusory search for literary success, however mundane it may be. But before we can toy with the idea of employing the auto-mechanism to suffuse every nook and corner of every single individual with the iridescent web of emotional archetypal semes[1] of unfathomable nuances, we need to experiment with the very possibility of instantiating those nuances through deft use of visual images, both still and moving, and establish the causational relationship that is deemed to be lurking somewhere out there in the day-to-day world. At least that is the premise upon which I will be weaving the threads of my essay as I look for the clues and manifestations of the confluences that could be posited to exist at this stage between the inner consciousness of the human psyche and the visual stimuli that surround it. I will employ digital data that in combination with sound elements can be easily presumed to produce the variegated nuances imaginable as amply instantiated in moving pictures. But my essay does not simply remain in the realm of movies/videos. I will introduce interactive programmed versions of the moving/still pictures to see how they in all or either of them with a slight difference in emphasis affect the person undergoing the visual experience. If the factors that induce multiple yet precalculated reactions on

[1] I am not specifically using this term in the manner many structuralists were fond of evoking it, namely that any semio/semantic constituents that clinch oppositional differences in many possible paired concepts. Rather, I am conceptualizing it more as semio/semantic units that constitute any larger meaningful conceptual aggregate such as expressed in literary work quoted in this essay. See an excellent exposition on the former example of the semes by Jonathan Culler in his *Structuralist Poetics* on pages 76-79, published by Cornell University Press in Ithaca, New York in 1986.

the part of the viewer are indeed found out or theorized in any usable form, that in its turn will be much utilized to bring the visual and human interactivity to a new level. With that in mind let us begin our search for that tenuous ground that is as yet filled with indefinable nebulous speculations. Before I suddenly plunge you into the bathetic practical technicalities, let me add that the program I often refer to in this essay is a mostly script dependent, proprietary one named Lingo that is rather widely known as of today, June 2002.

Since poetry excels in its evocative possibility, let me quote a sonnet by Shakespeare and argue for the imagistic potentiality of fusing between the text-based work and audio-visual elements. I would like to start by introducing Sonnet no. 35 by the same author.

No more be grieved at that which thou hast done;
Roses have thorns, and silver fountains mud;
Clouds and eclipses stain both moon and sun,
And loathsome canker lives in sweetest bud.
All men make faults, and even I in this,
Authorizing thy trespass with compare,
Myself corrupting, salving thy amiss,
Excusing thy sins more than thy sins are;
For to thy sensual fault I bring in sense
(Thy adverse party is thy advocate)
And 'gainst myself a lawful plea commence;
Such civil war is in my love and hate
That I an accessory needs must be
To that sweet thief which sourly robs from me.

This is indeed a complex of emotions that are deftly captured by one of the greatest poets of all times. The question is how we capture them objectively and render them in a form that is intelligible to all.[2] That may be a starting point. Although the ultimate purpose, as I mentioned above, is to trigger the gamut of those iridescent and floating emotions in a predictable way (or more precisely, to determine the causational linkage between the final emotions that are experienced by the maximum number of people and the factors that are directly involved in the mechanism), it is wise to focus on the

[2] Needless to say, I am not about to impose the absolute unseizability of the essence of the literary production as amply hinted at by Roland Barthes in his *S/Z*, a seminal work in the ad infinitum disseminability of language or its subcategory, literary work. If the object of my textual study is hopelessly "writerly," as Barthes says, then there is no chance that we can retrieve a fraction of the "meaning" of the work or fix even blurred image of the nucleus of the literary production. See *S/Z* by Roland Barthes, pp. 4-22, published by Hill and Wang of New York in 1986.

process that is implicitly embedded in the literary craftsmanship the present poet employs in the weaving of this particular poem. But even before that let us attempt to interpret the poem to come up with the general consensus as we try to colorate the concept found there with the apt images to buttress the senses to make the product even more vividly alive in the minds' eyes. Could the first line be a vain and valiant attempt to rationalize the pains inflicted upon the I that appears throughout the poem? But is the cause of such pains truly worthy of such sincere attention the subject seems to pay? That is a simple question that immediately arises in the reader's mind. Indeed as the second line confirms, or rather prevaricates, the psychology of the subject is complicated over the status of his relation to the other, be it his girlfriend or whatever adversarial position it happens to possess. Is the seeming determination expressed by the first line only that? Seeming and not in reality? Whatever is the case the ambivalence that is implicit throughout the poem is evidence enough that the subject is split over the suffering he himself has subjected to. On the conceptual level, however, one thing is certain. The two ideas put forth by the two clauses in the second line, "Roses have thorns," and "silver fountains [have] mud" contain truism. Anyone who has seen the plant designated by the name rose is aware that it is invariably endowed with thorns and any fountain, be it of silver or silvery white water (most likely reality forces us to concretize the latter unless one willingly allows himself to float in the Pegasusian world), unfortunately accumulates mud or dregs as a byproduct. By the same token both the moon and the sun are more often than not covered by multi-shaded clouds and occasionally hidden, either entirely or partially, by other planets. The following line continues in the same strain. Indeed no desirable object comes without its antithesis and often they accompany each other and there is no separating between he two in this mundane imperfect world. What do we make of this continuous duality/ambiguity expressed by the subject? Is it to alter or modulate the sentiment floated in the first line? Most likely yes. Otherwise the author would not have wasted ink over the random quatrain. That is another clue the reader is encouraged to seize upon to skim the poem on the objective conceptual level.

On another level we have to consider what is causing the author to dally between two swinging polar opposite sentiments. Is it a mere undesirability as to how he positions himself as to the implicit love he confesses he feels toward the lady? Or is it a complexity of feelings that constantly pull the author to sway through all gamuts of emotions? Or is it something else that lies deep yet interpretable as the active mind (by which I mean the reader) immerses himself in the world opened up before his mind's eye? In order to solve that question we need to proceed and continue absorbing all the variegated subtle nuances that emerge from the poem. Let us then go on to the next quatrain and see if we can find any evidence to persuade us to believe one way or another. The next line, "All men make faults, and even I in this," reads more like a concession in which the author allows himself a breathing space before venturing on to another intellectual ratiocination. The first half of the sentence is a generalization that is made to precede in order to include himself in that category. But whether it is a ploy to excuse himself or further castigate himself is not certain at this point. However, the second line gives us a clue as to where the drift of is his argument is moving. It is an accusation that he has been rather overindulgent when it comes to confronting his mistress's faults. He has been overlooking her violations of whatever kind by comparing them to others of larger magnitude (?) or de-emphasizing their significance to such a degree that he himself is even aware what moral consequences he is inviting. Now with that confession, "Myself corrupting, salving thy amiss," the author raises the issue he has been handling throughout the poem to a new level. If you look back at the first line, he started with a subjective self-suasion and then developed it into an objective arraying of factual statements that nevertheless brings out the issue of duality, or the inevitable coexistence of things with their adulterated counterparts. And now the argument has been brought to the moral and semi-religious level. According to his confession, he is corrupted and overindulgent because he looks away or overlooks the apparent faults she manifests. His overindulgence always surpasses the magnitude of the sins the lady commits, "Excusing thy sings more than thy sins are." However, what exacerbates his inner conflicts is his rational acceptance of his own fault. In other words, he admittedly knows the moral infraction the lady is making but he turns it into something forgivable in spite of himself. Or even worse he consciously bends his own moral integrity to accommodate the woman he confesses he cares with all his heart: "For to thy sensual fault I bring in sense." The most ludicrous result, again admittedly by the author, is that he is making himself the adversary in this curious case involving two parties. The parenthetical clause foregrounds that upside-down state ever so conspicuously except that the author turns the table and assumes a different perspective, placing the lady as the source of the viewing subject. The following line is a continuation of the litigious simile the author initiated before the parenthesis. By now he has completely framed himself and there is no way weaseling out of it. He needs to be (self-)interrogated and expose his precarious situation to the whole world. However, that is also intended from the beginning. The author has to bring out the ambiguous state he is forced himself in and by weaving all the intricate psychologically metonymic images in a series of quatrains he drops the contradicting yet concurrent emotional flows that occupy a mind that is as sensitive to the subtle shades of human emotions as he is. That is, by universalizing the very odd experience that is putatively only his own the author suddenly jots the objective confidence the reader has held so far. Whether one reads the final couple as a clownish surrender to the intricate emotional flows that barge on inside every human psyche or helpless compromise every lovesick tenderhearted yokel must make may be irrelevant. But what one has to recognize here is that the couplet, or for that matter the whole poem, is made up of kaleidoscopic emotional nuances that can possibly be divided into a multitude of colorlistic shades if the poem is to be rendered or helped by some kind of imagistic interpretation.[3] It may not be realistic either to assign one emotional shade to its supposed chromatic counterpart or group certain emotional and intellectual nuances into a set of scenic cues that are deemed to evoke a series of inchoate images that are in turn susceptible to appropriate emotional responses. But in the context of this essay I would like to remind the reader that a piece of artistic work such as the present one based on text is invariably pregnant with multitudes of meanings. The question then in the light of this essay is how to capture them and translate them into their closest imagistic entirety and used to evoke the original sentiment with added nuances. The task seems rather redundant and circuitous but if we need to program the artistically rich content and feed it to the intended audience we must arrive at the apt imagistic and colorational combinations to make the viewing experience truly worthy of the original poem and textual content.

First let us consider the possibility of giving rise to the programmed space in which the ambivalence that permeates the textual space is well reflected. Before we come up with the ultimate interface in which the viewer is actually come face to face with the pictorial rendition of the textual content we must lay out the foundation on which all such cosmetically satisfying façade must necessarily depends. Although the explications could become rather tedious, the reader is encouraged to follow them through. Since we are here primarily concerned with the graphical elements let us delve into the world of bitmaps in varying bit depths. Bit-depths may sound rather technical but they simply indicate the amount of information one can put in one bit of a group of pixels that constitute one pictorial element on the visible level. Because the more information one embeds in an image the larger the entire size of the file inevitably

[3] Perhaps some structural analysis exhibited long time ago by Roman Jakobson and his ilk can elucidate the underlining sophistication that buttresses the whole composition. If the subtle techniques used by the great poet can be analytically explained away there is a great chance that the interweaving of the kind I am advocating in this essay can be accomplished without much trouble. See *Language in Literature*, pp. 198-215, published by Harvard University Press in 1987.

becomes, it is paramount to think of a strategy before any concrete step is taken as to how many graphic units and of what depth one is going to include in one clip o presentation. Of course, the cons and pros of using one kind of depth as opposed to another depend on the over all picture the architect of the presentation has before the actual work starts. If it needs clear and absolutely photo-realistic movie throughout the presentation then the graphic elements are likely to be dominated by the kinds in the higher bit depth. But if on the other hand one intends to incorporate graphic images more or less as a background filler then no high-bit pictures may be required; doing so only takes up precious media space and most likely increases the loads on the presentation program in general. Needless to say, in the best of possible world even the background images are to be rendered as colorifically rich as possible all the time. But in the mundane and gritty world we must need to reside compromise is the albatross that hangs around our neck all the time. With that rather humbling admission let us consider the choices we have as to the bit depths of graphics we can incorporate into our projects using Macromedia Director.

The simplest and most primitive element we can put in the program is a 1-bit bitmap. The most primitive because the images saved in this depth has only a black or white coloristic state. Although the images of this kind can have only the monochromatic state, which is actually a state that is deprived of color, they are very small in size because of the amount of information they require. In addition to 1-bit bitmap one can use 4 bit bitmap in Director. Although 16 color rendition is much more colorful than the monochromatic one, it is still limited in its range of expressibility and realism. To circumvent that technical trap each 16 color can be customized on the Macintosh. That means if one wants to use a different set of colors it can be accommodated without resorting to higher resolution images. But the limitation is definitely there. Even though each 16 colors can be manipulated in a way that partially fulfill one's needs one cannot go beyond the number limit at any given time with a particular graphic. Besides varying color assignment to each bit of information is only possible on the Macintosh. On the Windows platform the 4-bit of chromatic information is always tied to one set of colors that is associated with the Microsoft VGA palette. What that implies is that if one intends to distribute his presentation cross-platform he is obliged to limit his colors to the Microsoft color palette. Otherwise all the precious colorations one has worked hours to produce will come to naught. Regardless of the platform one is working on, there is always that reality-impinged concept of limitation as long as one is concerned with distributing his final product to the maximum number of randomly chosen multitudes, which by the way is the norm and ideal toward which every developer should strive.

Thankfully for those who need more than 16 colors to express his ideas more color rich bitmaps are allowed in the director environment. The next in line is an 8-bit bitmap that contains 256 colors. Although once again it is not exactly photo realistically color rich, the number of colors the image of this kind contains is far numerous compared to the ones in the previous category. What is important is that the palette the images of the current class uses can have various ranges of grays, which conceptually fall between black and white. The graduated monochromatic colors enables the creator to produce various shades which in turn add realistic touches to the object he gives rise in the current environment. As an added advantage to the 8-bit mode is that both the Windows and the Macintosh platforms allow customization of the color palette that corresponds to the set of 256 colors. If one wants to change the chromatic nuances of the set of colors, therefore, one simply creates another palette or resorts to another preexisting one that dictates the representation of colors that are visible to the interfaced viewer. There is another choice when it comes to using different platelets in the Director's environment. If one wishes to use an image that is made up of a certain set of 256 colors and they are beyond the range of the already used palette, then one either decide whether to use the palette that is associated with the image or translate the chromatic set of the image through the current palette. Once a new palette is imported into the program it becomes a part of the entity that constitutes the whole presentation. It does not exist something other than or outside the environment of the Director presentation. The next color depth in the line of chromatic hierarchy is 16 bit. A 16-bit bitmap can have as many as 65,536 colors. The expressibity is quite astounding in this mode. The number of colors allowed in this bit depth exactly corresponds to that used in a normal television set. If one intends to distribute his presentation through that very media the encouraged bit depth to render the images may be 16. However, because the program has to be run on a computer, be it Windows or Macintosh based, the initial interface needs to be so robust and nimble as to render the images lightning fast as any set-top DVD player might do in its capacity as a generator/ decoder of clear and crisp moving images. Until the seamless interfacing of various media is achieved, the bit depth issue remains a truly nerve wracking choice for those developers who want to penetrate the wide finicky market with incalculably multitudes of tastes. The largest number of color choice can be achieved by incorporating 24/32 bit color images.[4] Needless to say, the available colors

[4] The notation here employed, 24/32, may seem rather odd. What it indicates is that the actual number of colors used on the monitor is the number expressed by the former in the set of figures

exceed 16 million in this format. But the vast array potentially projected onto your monitor goes far beyond that practically detectable by naked human eyes. Therefore, if one wants to produce a movie using all kinds of graphic elements needs to strike a balance between the practical technical issues confronting the program and the photo realistic presentation that is truly effective on the viewers. As with the previous color images the images rendered in this format do not have any direct linkage with sets of color palettes that are required in any other format.

Before going any further we need to understand the significance of palette. Although I have been making numerous references to that word ones without much experience in computer based graphical presentations may have a difficult time comprehending the function of a set of chromatic reference tables, which is what color palette signifies. Of course one can safely incorporate graphical elements without any understanding whatsoever of the concept. But having a firm grasp of the word helps in developing smooth and robust presentations that will most likely withstand almost all predictable situations. The parameters involved in successfully carry out computer-based presentations being so numerous, one cannot be too careful when building his products from the foundation on. Understanding color palettes certainly contributes to averting disastrous glitches which more than not happens when the computer is overloaded with graphically superb yet information heavy works. The caveat is, needs to say, it is only one of many variables that must be taken into consideration when putting together innumerable elements when preparing one's presentations. With that being firmly placed in the reader's memory, let us proceed with the explication of the concept that is on the agenda here. The program we are concerned here has some preset palettes that come with it as a default. Two of the palettes are each assigned to function as a default in their turn for the Windows and the Macintosh respectively. That is clear enough so far. But oddly enough and very troublesome to the programmers is that they do not exactly correspond to each other and the same value (by which I mean the location in the entire palette assigned to each platform as a default) does not necessarily produce the identical visible effect as the presentation is interfaced with the user. For instance, if one chooses to incorporate some 8-bit graphic elements then it is his ineluctable fate to choose an appropriate palette that is referenced by every single color in the graphic images. If the same color intended by the programmer shows differently on each platform, how is one to deal

represented and the remaining 8 bits represents the corresponding number of alpha channel information, which include rates of transparency, special effects and so on. See *Special Edition: Using Macromedia Director 8.5*, pp. 72-77, authored by Gary Rosenzweig.

with this non-trivial issue? The solution lies somewhere between the absolutist extremism and the relativist pragmatism. One way to deal with the issue would be to cover the entire screen so that no other fringe area of the desktop on the user's computer shows. If the entire movie/presentation takes over the computer then all the colors are displayed relative to the current user's palette, no matter what the original program designated or the original programmer intended to display the graphic elements in his presentation. If each pixel of color is shown in relation to the given color palette and as long as each one is appropriately associated to that palette there should be no visible incoherency before the viewer's eyes. Another caveat when one wants to display images in the 8-bit mode is not to mix different sets of palettes simultaneously. If two images try to reference themselves to two disparate color tables at the same time the computer/ program cannot handle it; as a result the program interprets the excessive colors by approximating the nearest colors to the ones that need to be displayed on the monitor. Needless to say, the outcome is most likely to be less than desirable, to say the least, and in most cases, completely unexpected by the editor/ programmer of the presentation. Therefore, unless the color shift that is predicted is allowable within the overall design of the presentation never use images that try to associate themselves with different set of palettes at he same time. Or even preferably, if the bit depth or the speed issue is not a problem, simply avoid the limiting format and adopt a higher depth presentation.

There are many more issues that need to be addressed before a convincing presentation is formulated. But for the moment let us leave the graphical aspect of the movie and move on to the textual element of the Director generated presentations. Since the direct linkage between the multimedia movie and a textual work has to be the text itself, the issue we will be looking at must necessarily occupy the central stage in any work that incorporates the bets the modern technology has to offer. Under Director textual elements are comprehensively referred to as strings. Therefore, I will be using that term frequently during my excursion into this subject, which is both primordial and up-to-date as far as the information technology is concerned. First, when we put together strings and let them constitute as a string each element of the string such as a, b, c and so on need to be concatenated in order to make up a meaningful whole. In fact, the simple rendition of "a" is a chunk of values that are associated with it; therefore, it in itself is made up of a set of bits of information strung together to constitute a certain signifier. If we look from a more macroscopic perspective, a set of characters need to be artificially strung together to make any sense to the program and ultimately to the viewer, who in this case constitutes one end of the interactive chain. Then how does one achieve that seamless coagulation of meanings through each discrete signal of semiotic bits? The following examples demonstrate some of the possibilities strings have in developing the semantic scope of the given presentation. If one wants to express certain ideas in text all he has to do is to put the chunk in quotation marks. A very common name such as John can be rendered as "John." That is nothing surprising except that the program needs to receive the semiotic signals through the insignificant seeming symbols of those quotation marks unless it is confused by the incoming information with the semiotic ingredients that are part of its vocabulary. If that happens the program either treats the command differently from the initial intent of the editor, which is very unlikely because of the limited number of program's vocabulary,[5] or it simply demurs and shuts itself off because it cannot process the incoming information due to the limited amount of leeway average computer programs usually are usually allowed to possess. But we must go into the actual mechanism involved in receiving and inputting information under the circumstances we are considering at this moment. For instance, as I mentioned already, simple information such as "hello!" can be fed in the following manner:

```
S = "hello"
Put s
```

The command in the second line returns the string "hello" without any surprise. However, daily contingencies more often than not requires to combine each semiotic and semantic unit with other myriads of strings that either modify or add the original meaning in every conceivable manner. How such feat[6] can be accomplished? By using concatenators and commands that allow combination of string chunks. Look at the following example. Suppose one desires to input "Hello, Tom!" and each word needs to be fed separately, how is it to be accomplished? It is programmed as follows:

```
S = "Hello, "
T = "Tom!"
U = S&T
```

[5] Although Director is constituted of hundreds and thousands of commands and symbols that are essential to achieve most of the interactive and presentational functions, it is easily eclipsed in the size of its vocabulary by any living language people daily use. As for the list of "words" Director employs, see *Macromedia Director 8 Shockwave Studio: Lingo Dictionary*, published by Macromedia. It comprehensively lists a usable vocabulary a programmer needs to make his presentations come alive.
[6] Stringing chunks of literal messages can indeed be considered a feat. Unlike daily combination of semantic/semiotic units in textual communication, computer-based communication has to receive every new bit of information in its defined manner and fusing one with the other brings in an entirely new procedural operation before any overall meaning can be arrived at. Therefore, it is not a matter of incorporating redundant subtle nuances, which may be introduced as the semantic/semiotic complications increase, as in daily communications, but rerouting the information signal through the "semiotic circuit" all over again when the computer-based machines take in new input.

```
Put U
-- "Hello, Tom!"
```

Now, notice that I inserted a space after the comma in line one, which is preceded by the variable S. That is the reason why the command put U results in the manner noted above. But leaving a space after a letter and before a closing quotation mark becomes a little confusing, especially in complicated programming where bulky expressions jostle with each other. Rather than allowing such very likely confusion to wreck the fruit of lucubration, why not use an alternate method, which is much more explicit and thus likely to reduce careless mistakes to a great degree? If one has gone through with the process of checking lengthy expressions and has been troubled with the nagging literal input that did not appear as it should have been, then he must surely appreciate the importance of explicit statement. The alternate and more powerful description of the above statement goes as follows:

```
S = "Hello,"
T = "Tom!"
U = S&&T
Put U
-- "Hello, Tom!"
```

Admittedly there is not much difference between the above and the one preceding it. The only difference is that the space-inserting concatenator, indicated by two ampersands, has been used to elicit the right combination of two chunks of strings. In case you want to insert letters at a certain location within already exiting chunks, methods are not so complicated. Let us try the following.

```
Put u
-- "Hello, Tom!"
put "h" after u.char [8]
put u
-- "Hello, Thom!"
put "as" before u.char [12]
put u
-- "Hello, Thomas!"
```

Needless to say, we could have used the first command to accomplish the same effect. Or alternately we could have used the following command.

```
Put "Scottino" into u.char [14]
Put u
-- "Hello, Thomas Scottino!"
```

That is a perfectly intelligible sentence. Of course, we might as well skip the whole command and orally input all the necessary addendum and information into the computer memory and have that passed to the interfacial stage. But unfortunately programs of the kind we are dealing with need the vocabulary to perform the operations we want to them execute. At this stage, that is a necessary detour, which nevertheless saves us a lot of extra work in the end. Now, what about substitution of strings? How is it to be accomplished? That is not complicated either. Essentially it works in the same manner. Consider the following.

```
Put "Lucia" into u.char [8..13]
Put u
-- "Hello, Lucia Scottino!"
```

The operation is quite simple. First you set the range of characters to replace in the brackets and command the program to insert a given string into that particular slot. The result is a clean substitution that replaces the original chunk with the desired one. What is noteworthy here is that the target string and the substitute string may not be the same size. Note that the alternate string "Lucia" does not correspond to the original chunk "Thomas" in the number of letters it contains. The former is obviously one short of the latter. This trivial feature of the series of command I cited here makes string operation very flexible. You almost execute it without any sense of what takes place on the basic programmatic level. (There is certainly no need to pay attention to memory allocation, which could complicate the editing work considerably although it may contribute to the interfacial efficiency.)

There are many other useful string operations that are feasible in Director. Look at the following example.

```
on allCaps text
repeat with i = 1 to text.length
thischar = charToNum (text.char[i])
If thischar >= charToNum("a") and thisChar<= charToNum("z") then
thischar = thischar – 32
put numToChar(thischar) into text.char[i]
end if
end repeat
return text
end
```

This is a command culled from Gary Rosenzweig's *Using Macromedia Director 8.5* (p.326) that converts lower case letters to their uppercase counterpart. What it does in the first line is to take a parameter from the incoming source, here represented by text. (Needless to say, it does not have to be called as such; the name text is merely a convenient metonymy for all that is coming in as a string.) Once the function receives the information it subjects it to a methodical scrutiny by checking every single component of the chunk from 1 (the starting letter) to the final letter (here indicated by text.length). Each letter in its turn is represented by the variable i[7] that is placed on the left side of the equation. Once the methodical check starts each letter in a given permutational spot is converted to the numeric value it holds in the program. The value then is safely stored in a variable thisChar. The next line in the function ascertains if the given datum is indeed a lower letter within the range between a and z. The process utilizes the fact that lower letters between a and z fall within a defined numeric values and all those which do not excluded from further consideration at this stage. Once the datum passes beyond the conditional barrier, it is transformed to its uppercase counterpart by the subtraction of numeric value 32. It is possible because every lowercase letter has its uppercase counterpart exactly 32 numeric value smaller than it is. Thus, the variable thischar, which is on the left side of the programmatic equation, comes to hold the desired numeric value in line 5. The only thing left for this function to accomplish is to put the corresponding letter of the numeric value in the right spot in the original order of the string elements. The numToChar( ) operation does the conversion and the put operation accomplishes the second in line 6. It is a simple handler but executes the command quickly and without any demurral.

Let us apply the power the allCaps handler gives us to the poem we started this essay with. Suppose we need to emphasize a chunk of words/string in the poem. Or more concretely, the first four words in line one, "No more be grieved at that which thou hast done." The command need to have the handler to execute the required work is allCaps("No more be grieved"). It seems very simple but all that is required of the editor is exactly that after the target handler has been constructed. But unfortunately, the command does not print out the result of the call to the handler. In order for the editor to see the feedback he needs to store the outcome and have it utilized by the program in such a manner that the chain reaction that derives from the initial execution ends up with the final capitalization of the original string. But as a shortcut let us view the outcome without the usual trappings of lines that intervene between the

[7] The i stands for index, obviously. Although it does not have to be i(in fact any letter can assume the role of a variable in Lingo unless it already has a designated role in it), every programmer is advised to use letters in a semio-semantically significant manner. That will eventually save him much trouble.

initial feed of the information and the final result, which would be the norm in real life programming than otherwise. The immediate feedback can be obtained by the following simple command.

```
Put allCaps("No more be grieved")
-- "NO MORE BE GRIEVED"
```

Interestingly enough there are all kinds of operations that can be executed to modify and arrange the chunks of words in the original poem by resorting to the Lingo's vocabulary. See the following.

```
S = "Clouds and eclipses stain both moon and sun"
Put s.word.count
--8
put "[there is always something that adulterates the purity of existence.
Meteorological phenomena brought in by the poet support that major premise.]" after
s.word[8]
t = s.word [1]
allCaps (t)
u = s.word [3]
allCaps (u)
put s
-- "CLOUDS and ECLIPSES stain both moon and sun [there is always something that
adulterates the purity of existence. Meteorological phenomena brought in by the poet
support that major premise.]
```

Here we have achieved a minor annotation of the poem while keeping the sight well focused on the artistic work itself. Of course, all the preceding experiments are just an attempt to demonstrate the interactive possibilities using the simple handler we have constructed above (plus numerous functions that are a part of the Lingo by default). Granted that the interpolation of the kind that accompanied the foray into the interactive presentation of Shakespeare's artistic work should not be made frivolously. But the ramifications such intrusion into a major citadel of literature spawns seems to justify rather presumably haphazard programming attempt to come up with visualizing conceptual treasures. Back to our programming quest. If one wants to read the annotation of a certain line in the poem, all you have to do is to request or command

```
S = "The entire poem in string"
Put s.line[x].word[y .. z]
-- "[feedback of all the string annotation that is embedded in the poem]"
```

The mechanism seems rather primitive but after discreet development in the intended line the corpus of the whole presentation is sure to impart significant amount of new insights into the target work. Be that as it may, string commands are an ingredient that cannot be ignored by somebody who is seeking to build a robust and yet meaningful interface with an artistic expression using the computer supported platform.

Now it is time to look at the sound related issues that are likely to become the focal point in trying to bring the multimedia presentations to their fullest potential. There are, like other features in the program, hundreds if not thousands of functions that are possible to construct based on the default vocabulary embedded in Director. Needless to say, it is impossible to cover all of them. We will look at some of the more useful and interesting ones that are likely to be fitted within a purview of a small number of pages available at the moment. There is a property named pan that determines the way sound is played more or less on one side of the speaker than the other. Suppose an object position controls the way the sound plays and the object incessantly moves around the stage as time passes. The following lines meet such requirement.

```
pSound = "grvySound"
On beginSprite me
Sound(2).queue(pSound)
Sound(2).play( )
End
On exitFrame me
X = sprite(me.spriteNum).locH
StageWidth = (the stage).drawRect.width
cL = 200*x/stageWidth – 100
sound(2).pan = cL
end
```

The channel used for the sound is arbitrary. It could be any number between 1 and 8. The function queue( ) loads the designated sound and readies for the next command. The play( ) command actually triggers the sound to play. Therefore, in the first handler the target sound starts playing as soon as the stage is drawn on the user's monitor. The fourth line in the next handler calculates the relative value of the pan in relation to the stage width of the presentation. The number in the numerator reflects the panning values assigned to the pan property: 100 forces the sound come only out of the right speaker and conversely –100 out of the left speaker. The actual plotting of the sound is executed by the proceeding line. If we apply this simple mechanism to the creation of the interactive interface some interesting effects can be accomplished. For instance, if lines 5 to 8 are associated with some moving object on the stage, be it a concrete representation of the hinted object of love or an abstract rendition of such emotional stage of the poet (or the pretended I in the poem) fresh angles cold be posited to look into the psychological state of the lover or the relationship depicted here as a whole. Let me quote the lines to refresh our memory.

All men make faults, and even I in this,
Authorizing thy trespass with compare,
Myself corrupting, salving thy amiss,
Excusing thy sins more than thy sins are;

Suppose the sound that is playing, as if it is truly playing on the stage, shifts its pan ever so restlessly while the reader's consciousness is fixed on the first line, what would be the added effect on his overall experience? The line states the aberrant judgment one makes in life, especially vis-à-vis one's lovers. It is a fit metaphoric landscape for wavering pan. What if a tune that is susceptible to ever so slight whiff of air on the wind instrument changes its panning value while the tune itself remains uncertain in its quality about which way it will resolve? That music backdrop is sure to corroborate the moral dithering the author is confessing to be having whatever its true nature may be. The first line almost makes sure that the reader perceives the willful nature of the author's aberration from his sense of moral integrity and right judgment as he simultaneously hints at self-flagellation and excuse. The intentional guilt is further brought to the fore by the active authorization the I grants each instance of the trespass the other makes. The purported authorization is made more explicit by the term compare because the act is completely volitional and more than a little dependent upon the rational part of the author's ratiocinative faculty. Imagine once again the sound/music is constantly panning between right and left or vice versa as the object associated with the psychological texture of the given lines oscillates on the stage. The comparative act is reiterated as the lines progress. While the author tries to protect the integrity, be it moral or personal depending on the mental picture he depicts of his loved partner, of the target of his love, he inevitably brings himself down in each of those aspects he is judging the other in to the extent that he "corrupts" himself while compensating the other's failures with his own overly indulgent attitudes. The relations depicted here is like two points constantly ascertaining their positions in respect to the other, except that in the subject's case he is the one that is always shifting his psychological locale in an obvious attempt to morally over-compensate. The music continually oscillates from moment to moment, which in time should correspond to psychological oscillation the reader perceives the author is experiencing.

Sound elements can be utilized in many other ways. Another possibility to effectively integrate interactive features computer based presentations generate and the textual corpus is to use the following method.

```
On cuePassed me, chnnlNum, cueNum, cueName
Sound(1).queue (cueName)
Sound(1).play( )
End
```

Of course the playback head needs to be stationary for the program to be able to intercept the message, here represented by cueName. The handler to accomplish that is state as follows.

```
On exitFrame me
Go the frame
End
```

The series of functions are quite simple but nonetheless the combined effect on the whole presentation can be enormous. Suppose a context sensitive music plays while the poem is being displayed when a selected line is suddenly read out to grab the attention of the viewer on cue. The added impact that turn of events has on the viewer would be enormous. The present mechanism not only helps enhance the overall ambience of the presentation but in combination with the string-based interactions sound cue mechanism is sure to deepen the viewer's appreciation of the poem as a whole. Not only does the basic programming delineated here control the timing and the order of sound effects that play out before the user but also render itself to modulating various elements of those sound clips being utilized in the given context. As one possible diversification from the main model given above, let us consider those aspects that deal with the pitch modulation. Consider the following coding.

```
CrrntPitch = random (8)
Sound(3).queue( [#member: member("accmpMsc1"), #rateShift: CrrntPitch] )
Sound(3).play( )
```

As expected, the music designated by the queue function plays according to the rateshift value determined by the random function. Since the rateShift[8] is to be decided every time the random function is run, there is an unexpected surprise on the part of the

[8] The rateshift is a factor that influences the way a given piece of music/sound plays. Because the faster a sound clip plays under the same condition, the higher in pitch it is likely to be, the values, which are spread within the range of 1 to 8, the optimum speed and the pitch are those which are initially recorded or sampled. But in a demanding environment such as a computer-based multimedia presentation, utilization of one feature or the other in an attempt to realize the maximal potential in terms of conveying the underlying ideational gestalt may as well be justified and be much taken advantage of no matter what the cost, be it fidelity or acoustic satisfaction.

recipient of the content as the music plays somehow modulated every time he goes over the same passage (although admittedly there is a chance that the same pitch is repeated every once in a while.) Let us combine the time trigger scheme shown above and the current pitch modulation mechanism to produce a setup that is more elaborate than if any one of them is used independently. Suppose that as the reader's eye has reached the end of the line, "For to thy sensual fault I bring in sense," a timing object initiates a series of acoustic flourishes. The timing object can be created in the following manner.

```
RecTimer = new(script "timerMatrix", the ticks)
RunTimer(recTimer)
```

Needless to say, the timer should be triggered so many seconds before the approximate moment when the reader is supposed to have finished reading the target line. The following coding sates the general stream of commands and input interceptions needed to accomplish the textual and sound synergy I have been delineating. Suppose the presentation takes place while the playback head remains stationary.[9]

```
on exitFrame me
  go the frame
end
```

This is clear and simple. As the playback head tries to move onto to the next frame the command, go (to) the (same) frame, sends it back to the beginning of the same frame. The plan is that while the playback head remains in the same frame ambient or (con)text appropriate music, which has been started somewhere in the presentation, continues playing. In the meanwhile the reader's eyes trace the markings on the monitor which reflect the artistic work produced by the initial author in ever varying degrees. Bu the decisive moment arrives when the recalculated juncture arrives when the attentive eyes have presumably reached the end of the given line and the coded script intercept the message that has been embedded in the playing music. The interceptive process can be encoded as follows.

```
on cuePassed me, chnnlNum, cueNum, cueName
  if cueName contains "strtTimer" then
    timerObj=new(script "timerPrnt", the ticks, "crrntRct")
    runtimer(timerObj)
```

[9] The stationary playback head does not mean the presentation itself remains static. While the playback head is securely moored in one particular frame, there are thousands of things occurring on the stage. Presenting moving pictures, videos and interactive animations are only a few of the representative contents that can be incorporated into the Director-based movies. For more on the technical issue of the playback head, see *Macromedia Director 7 and Lingo Authorized*, pp. 14-118 by Phil Gross, published in 1999.

```
  end if
end
```

Here the cueName corresponds to the message pre-inscribed to the original music at a certain critical moment the programmer deemed fit for the eventual textual-music interactive fusion. Once the message has been received by the current handler it initiates a series of actions in which a timer object is created and the timer—that has come into existence as a result of the interaction between the commanding script and the recipient parent script—is set in motion. Now it is high time that we saw the recipient script that answers the call from the coding we cited above. The script—the type of which is classified as parent in the taxonomy of the program we are dealing with—begins with the object creating handler "on new me." Of course all the parametric properties need to be declared somewhere in the script, although the beginning and on top of everything else is usually a preferred location. I will follow that time-honored custom.

```
property pTicks, pCrrntRct
on new me, crrntTime, crrntRct
  pTicks=crrntTime
  pCrrntRct=crrntRct
  return me
end
```

The parameters receive and hand over the corresponding arguments sent from the calling script. Each one is stored in the respective properties that are placed on the left hand side of the equation. Here pTicks holds the value represented by crrntTime, which in turn—if you see the calling handler—takes after the ticks[10] the computer sends to the program. The command preceded by "return me" effectively returns the values or the object produced by the current operation to the calling handler, allowing the calling side to manipulate the values independently held by the parent script. Because the newly created object is reachable from the calling side the second command in the on cuePassed handler sets off the timer mechanism embedded in the corresponding parent script.

```
on runTimer me
  if (the actorList).getOne(me)=0 then
```

[10] The ticks are the continuous stream of pulses, as it were, that a given computer generates from the moment it was turned on. Because the concept represents a ceaseless stream of signals, it can be utilized in various ways to measure a given duration of time. Each tick signifies 1/60 of a second, or conversely, 60 ticks amounts to one second. By comparing the ticks at any given moment to those at some other moment and subtracting the former from the latter one can arrive at the time elapsed between those two segments while the Director driven presentation is going on.

```
    (the actorList).append(me)
  end if
end
```

Actually, the time elapsed will be calculated by subtracting the number of ticks pulsed after the previous ticks measured when the moment the calling script was evoked. The difference between the two is expressed as the comparative values between the left and the right terms in the following coding.

```
on stepFrame me
  if the ticks<pTicks + 60*5 then
    exit
  else
    puppetSound 1, 0
    sound(2).queue(member (pCrrntRct))
    sound(2).play()
    (the actorList).deleteOne(me)
    go to "rctPhs1"
  end if
end
```

In line two of the above handler 60*5 indicates the surplus value beyond the ticks that was measured at one point in the flow of presentation. Since the ticks on the left continuously measure the time elapsed since the computer was turned one, the right term, the number of ticks at pint X plus an arbitrary accumulation of numbers decided by the programmer, easily adjust the timing fro whatever purpose the timer object has been initiated in the first place. In this case, as I have already mentioned, the timer is turned on to first wait and then sound the recitation sound effect. Here the channel used to sound the target clip is different from the one used for the previous ambient music. Because the music playing in channel 1 is turned off before the current one is to be played, channel switching does not have much significance. But considering all the contingencies that might come in the way of the editor, such as playing sound FX simultaneously in two diverse channels, although that is not necessarily a recommended strategy for the reason that each clip could interfere with each other, the present coding could be justified.[11] Here as before the sound element is first loaded—

[11] Although it might not be the case in many cases because brevity and clarity of scripting often helps editor to build more solid structures and the edifice that will eventually arise from them, leaving numerous options later on to develop other ramifying codes to deal with interactive contingencies, for example, more often than not prove more sagacious than otherwise. The seemingly redundant coding on the surface may not always be just an ineffectual and callow attempt on the part of the editor to

sound(2).queue(member (pCrrntRct))—to the program before it is actually played. The final two lines in the current handler first eliminate the object from the actorList[12] and then move the playback head to the destination appropriate from the next scene development. That is to say, a scene that is to synergistically play out the essential artistic elements together with the music FX, in this case as I have numerously adumbrated a sound clip with the recitation of the line where the viewer's eyes are presumed to be fixed.

Using the timer object strategy there are many things possible to evoke interest in the artistic work the reader is facing at the moment. The straightforward one enumerated above is one. Triggering timers at various points in the reader's immersion in the aesthetic world is another. For instance. When the reader's eyes have supposed to have traced the lines auditorily enhanced by the interactive presentation the program then transitions in another line, either posterior or anterior in its relative position to the line just dealt with, for the reader perusal. And once again when the subject's eyes have reached, that is presumed to have, the end of the given line the timer that has been started beforehand brings his attention to the next line, which may or may not have to be immediately following the line in question, either by the sound or visual effects. Scattering all the preceding tricks throughout the target work will have a kind of discontinuous effects on the working mind of the subject that the overall mechanism is sure to thrust new insights into the viewer's ken which he might have overlooked in spite of himself. Multimedia-based presentations are in that sense conducive to new perspectives on the works that may have become "common" through their currency in today's information oriented society. Although that might remind many readers of the Romantic ethos deliberated upon by none other than the Master of Romantic poetry William Wordsworth,[13] exposing artistic works in iridescent

---

desperately interactivate his presentations.

[12] The actorList is set up as a global list, a list that can be accessed from any place in the presentation, and putting an object in the actorList makes the object susceptible to the signal named stepFrame. Once an object is in the actorList, therefore, it can be manipulated to go through the ticks operation explained above and many other macroscopic processes, programmatically speaking. That is why appending to and deleting the object from the named list allows the editor controlling timing from any locale in the presentation, as demonstrated above.

[13] The sentiment most conspicuously expressed by the same author in this context appears in a prolegomenon entitled "Preface to Lyrical Ballads." Succinctly summarized as making ordinary things strange, the estranging strategy sounds both new and old in the same way as the multimedia reinterpretation of the Shakespeare's poetic work does.

> The principal object, then, which I proposed to myself in these poems was to choose incidents and situations from common life, and to relate or describe them, throughout, as far as was possible, in a selection of language really used by men; and, at the same time, to throw over them a certain colouring of imagination, whereby ordinary things should be presented to the mind in an unusual way; and, further, and above all, to make these incidents and situations interesting by tracing in the, truly though not ostentatiously, the primary laws of our nature...." (Quoted from *The Norton*

lights by dint of the new technology is both something new and old in that the technology utilized in the kind of presentations I have been describing has become available only recently and the attempts to commingle the current methodology with the old seemingly discrepant contents have been tried and retired since the dawn of civilization. Before we become mired in the philosophical aspect of the attempt I am describing, let us detach ourselves once more from the deeper issues and see how the actual multi timer triggers can be set and pulled. Look at the following lines.

Let me confess that we two must be twain
Although our undivided loves are one:
So shall those blots that do with me remain,
Without thy help by me be borne alone.

This is the first quatrain of Shakespeare's sonnet 36.[14] The following codes are set to deliver the quatrain in the most effective manner possible in the digital environment we have been envisioning. But rather than simply reiterating the previous strategy, in which the simple timing mechanism starts calculating and trigger a given series of actions, let me introduce you to a more elaborate manner of presenting the above lines. First, let us consider a situation where consecutive pictures are shown to the viewer one after another in an order that is deemed to enhance the viewing experience of the audience and crescendo, as it were, the ambient mood that is appropriate for the movie as a whole. Look at the following proposition.

```
Property pWaitT
On beginSprite me
StartTimer
pWaitT=5
End
On exitFrame me
If the timer<60*pWaitT then
Go the frame
Else
Go next
End if
end
```

Anthology of English Literature, vol. 2, pp. 142-143, published in 1993.)

[14] The numbering used here relies on *William Shakespeare: The Complete Works* edited by Alfred Harbage, published by The Viking Press in 1975. Needless to say, as one of the editors overtly admits, spelling and punctuation are modernized to convey accurate imageries and nuances to the modern audience.

The two handlers are to be executed, of course, after the movie has actually started. Therefore, before these commands are issued, there is an unlimited number of possibilities that can in turn trigger various interactive actions for the diversion of the audience. But that segment is to be skipped for the moment to make my structural design clearer. Back to the handlers stated above. The first property pWaitT is the value that is to be determined according to the needs of the editor. It controls the length of time a given picture is shown to the audience. The value given in the beginSprite handler happens to be 5 but it can be varied according to the kind of effect the programmer is seeking. To make the coding more flexible and at the same time more accessible even to the novice, another handler can be utilized to fill in the value represented by the same property. The handler for that purpose can be written as follows.

```
Property pWaitT
on getPropertyDescriptionList me
  pList=[:]
  pList.addProp(#pWaitT, [#comment: "Waiting Time in seconds", #range: [#max: 10, #min: 2],#format:
#integer, #default: 5])
  return pList
end
```

The part within the parentheses, preceded by the symbol #range, can be deleted. But by setting up the range, you can generate a graphic user interface for the individual editor to manipulate the time for the duration of which the target picture can be perused. In any case if the getPropertyDescriptionList handler is used the person editing or constructing the presentation does not have to enter inside the coding. All he or she has to do to initialize the variable is to attach the script that has the named handler as part of it to any object or location where the value is utilized. For instance, if the pWaitT is to determined the length of time a given picture is shown and at the end of which duration to send the playback head to the next destination to let the program display the ensuing picture the getPropertyDescriptionList handler not only becomes an integral part of the preceding script but also necessitates deletion of line that sets a value to the property pWaitT. Now the first graphical presentation is set up according to the blue print, let us consider embellishing the movie even further with some sound elements. What if the transition from scene A to B (in this case change in graphical elements) is accompanied by music or at least some sound effects? How is it to be achieved? Just insert the following commands before sending the playback head off to the next label within the exitFrame handler above.

```
puppetTransition 47, 2, 1
sound(2).queue([#member: member("pict2")])
sound(2).play()
```

Needless to say, the first line in the modification is to add more interest to the presentation. It forces the program to play a transitional animation between the given scenes. All the arguments represented by numbers each control the type, duration and response to the discrepant elements in the two scenes, respectively. Besides the first argument appended to the queue function, you can add another called rateShift, which modulates the pitch of the sound element to be played by the proceeding function play( ). If, for instance, the sound needs to be played at a higher pitch than its original one, you can rewrite the line in question in the following manner.

```
sound(2).queue([#member: member("pict2"), #rateShift: 8])
```

Of course, as previously stated, the given method simply plays the sound clip at a faster rate. The result may not be satisfactory if the purpose sought by the editor is to modulate only the pitch and nothing else. Along with the pitch the duration is inevitably affected. But other than that there is not much noticeable defects to the method used here. As long as the major objective is accomplished the kind of inconvenience experienced is quite negligible and something that is easily overlooked. Now let us make the values and other parameters more easily settable and accessible to the editors. If one takes recourse to the getPropertyDescription handler, which we already did above, the objective is easily accomplished. See the following execution.

```
property pWaitT, pTrnsType, pPitch, pSndNm
on getPropertyDescriptionList me
  pList=[:]
  pList.addProp(#pWaitT, [#comment: "Waiting Time in seconds", #range: [#max: 10, #min: 2],#format:
#integer, #default: 5])
  pList.addProp(#pTrnsType, [#comment: "Transition Type", #range: [#max: 52, #min: 1], #format:
#integer, #default: 1])
  pList.addProp(#pPitch, [#comment: "Sound Element Pitch", #range: [#max: 12, #min: -12], #format:
#integer, #default: 1])
  pList.addProp(#pSndNm, [#comment: "Sound Name Used for Transition", #format: #string,
#default: ""])
  return pList
end
```

Here the noteworthy feature is that each numeric value is to be set through a graphic user interface to facilitate quick initialization of respective variables. An extra feature makes the getPropertyDescriptionList handler so usable is that the same script can be used to allow different sets of variables for each frame the script is assigned to. For instance, if one wants to play a different music clip, which is very likely if the sound element is to enhance the viewing pleasure of the audience as they watch different graphic images appear and disappear from their ken, one just enter the target sound name in the dialogue box that is part of the interface units in the given frame script. Therefore, the same handler leads to the economy of scripting, which is definitely an advantage if one is thinking of writing a substantially long corpus of codes.

Let us think of a situation where the intended scenes develop not only with sound elements alone but other more literally charged elements such as text. Suppose textual clips also fade in and out of the scenes in tandem yet somewhat independent of the sound elements. What are the possibilities? In order to accomplish that one needs a timer object, the kind used above. Look at the following pair of scripts.

```
property pWaitT, pTrnsType, pPitch, pSndNm, pTxSprt, pTxSpMem, pTmObj, pWtTmTxtChng,
pFrstTxtStrng, pScndTxtStrng, pThrdTxtStrng, pFrthTxtStrng, pTxTrnsType
on getPropertyDescriptionList me
  pList=[:]
  pList.addProp(#pWaitT, [#comment: "Waiting Time for Scene Change in seconds", #range: [#max: 40,
#min: 20],#format: #integer, #default: 20])
  pList.addProp(#pWtTmTxtChng, [#comment: "Waiting Time for Text Change", #range: [#max: 10,
#min: 4], #format: #integer, #default: 5])
  pList.addProp(#pTrnsType, [#comment: "Scene Transition Type", #range: [#max: 52, #min: 1],
#format: #integer, #default: 1])
  pList.addProp(#pPitch, [#comment: "Sound Element Pitch", #range: [#max: 12, #min: -12], #format:
#integer, #default: 1])
  pList.addProp(#pSndNm, [#comment: "Sound Name Used for Transition", #format: #string,
#default: ""])
  pList.addProp(#pTxSprt, [#comment: "Text Sprite Number", #range: [#max: 20, #min: 10], #format:
#integer, #default: 10])
  pList.addProp(#pTxSpMem, [#comment: "TextSprite Member", #format: #string, #default: "fxTxt"])
  pList.addProp(#pFrstTxtStrng, [#comment: "First Text to Appear", #format: #string, #default: ""])
  pList.addProp(#pScndTxtStrng, [#comment: "Second Text to Appear", #format: #string, #default: ""])
  pList.addProp(#pThrdTxtStrng, [#comment: "Third Text to Appear", #format: #string, #default: ""])
  pList.addProp(#pFrthTxtStrng, [#comment: "Fourth Text to Appear", #format: #string, #default: ""])
  pList.addProp(#pTxTrnsType, [#comment: "Text FX Transition Type", #range: [#max: 52, #min: 1],
#format: #integer, #default: 52])
```

```
  return pList
end
on beginSprite me
  sprite(pTxSprt).puppet=1
  sprite(pTxSprt).ink=36
  sprite(pTxSprt).member=member(pTxSpMem)
  sprite(pTxSprt).loc=point(random(500)+150, random(400)+100)
  member(pTxSpMem).font="century"
  member(pTxSpMem).fontSize=25
  member(pTxSpMem).color=rgb("#FF6600")
  member(pTxSpMem).text=pFrstTxtStrng
  pTmObj=new(script   "timerPrnt",pTxSpMem,   pTxSprt,   pWtTmTxtChng,   pScndTxtStrng,
pThrdTxtStrng, pFrthTxtStrng, pTxTrnsType )
  runTimer(pTmObj)
  startTimer
end
on exitFrame me
  if the timer<60*pWaitT then
    go the frame
  else
    puppetTransition pTrnsType, 2, 1
    sound(2).queue([#member: member(pSndNm), #rateShift: pPitch])
    sound(2).play()
    go next
  end if
end

property  pSpNum,  pRcvTm,  pWtTime,  pMemNm,  pTxTrnCntr,  pScndTx,  pThrdTx,  pFrthTx,
pTxTrnsType
on new me, memNm,txSprt, wtTime, scndTx, thrdTx, frthTx, txTrnsType
  pMemNm=memNm
  pSpNum=txSprt
  pRcvTm=the ticks
  pWtTime=wtTime
  pTxTrnCntr=1
  pScndTx=scndTx
  pThrdTx=thrdTx
  pFrthTx=frthTx
  pTxTrnsType=txTrnsType
  return me
end
on runTimer me
  if (the actorList).getOne(me)=0 then
    (the actorList).append(me)
  end if
end
on stepFrame me
  if the ticks<pRcvTm+ 60*pWtTime then
    exit
  else
    pTxTrnCntr=pTxTrnCntr+1
    sprite(pSpNum).member=member(pMemNm)
    sprite(pSpNum).ink=36
    sprite(pSpNum).loc=point(random(400)+150, random(400)+100)
    member(pMemNm).font="century"
    member(pMemNm).fontSize=25
    member(pMemNm).color=rgb("#FF6600")
    if pTxTrnCntr=2 then
      puppetTransition pTxTrnsType, 4, 15, 0
      member(pMemNm).text=pScndTx
      pRcvTm=the ticks
    else if pTxTrnCntr=3 then
      puppetTransition pTxTrnsType, 4, 15, 0
      member(pMemNm).text=pThrdTx
      pRcvTm=the ticks
    else if pTxTrnCntr=4 then
      puppetTransition pTxTrnsType, 4, 15, 0
      member(pMemNm).text=pFrthTx
    else if pTxTrnCntr=5 then
            (the actorList).deleteOne(me)
    end if
  end if
```

```
end
on xOut me
  if (the actorList).getOne(me) then
    (the actorList).deleteOne(me)
  end if
end
```

I admit the pairs are rather complicated and lengthy scripting that may not be easy to grasp intuitively. But they hold the key to timing the appearance of various elements that are to play important roles in the presentation. In this case the visual effect is accomplished by the coming in and out of the text that constitutes the part of the target work. Without further delay let me explicate the inner mechanism involved in accomplishing the desired end through the coding introduced above. First all the properties/variables are to be initialized through the graphic user interface to simplify the process. That is taken care of by the getPropertyDescriptionList handler. As the editor tries assign the pertinent script to a desired location all the dialogue boxes appear asking him to fill in appropriate values. Unless he does, all the default values are used, preventing uninitialized variables, which inevitably cause the program to shuts down. Once the presentation gets under way, the beginSprite handler in the same script accepts the namesake message and causes all the commands incorporated within the handler to be executed. What it does is, first, puppet the designated channel (here represented by variable pTxSprt which has been effectively initialized before the program has run), then set the ink of the sprite to 36, which is transparent other than the colored element, and populate the channel with the desired member (the content of the pTxSpMem has been determined already by the previous handler) and sets the location of the same sprite (which is randomized to add some surprising element to the presentation by the random ( ) function). The font of the text that is to appear in the puppeted sprite is set in the next line to "century" and along with the font its size and color are fixed by the proceeding two lines. The actual text that will inhabit the text element is decided by the next initialization. Because the string variable pFrstTxtStrng is already determined in the previous handler (provided that the editor did input the necessarily information; otherwise the variable remains uninitialized and the object initialization fails) the text member possesses the given string value. At this point the poetic line (once again suppose the above quatrain was the elements editor was dealing with here) appears according to the dictate of the editor incorporated within the current handler, i.e., somewhere within the demarcated area of the stage.

Once the string value is set and populated in the target text member, the next couple lines generate the timer object that will calculate the number of seconds to wait before inserting the next text element into the same member until all the target lines are displayed and the playback head is effectually sent to the next segment of the presentation. But in order to fully understand the series of processes leading to the elimination of the functionality of the timer object by its removal from the actorList, it is essential to see the inner workings of the parent script called by the behavior script that has been introduced above. As is evident from the number of the arguments the first handler, on new me, of the parent script takes, the present script relies heavily on the values passed from the calling script. That in turn translates into the high flexibility the whole corpus of presentation is endowed with because all the editor needs to if he desires to modify string elements of the parametric values is to access the graphic user interface and input the necessary modifications. The time saved and the intellectual exertion spared through the simple mechanism is enormous. The more complex the entire movie becomes, the more energy is likely to be saved, both physical and mental. Note that the object created within the new handler is safely returned by the final command return me. The simple command makes the time object available for further used in the calling script, which otherwise lacks the handle to deal with the timer any more. The actual ticking of the timer, as it were, does not start until the runtimer( ) command is issued in the preceding script. The mechanism is very simple. The lines within the runTimer handler first checks if the object me is already in the actorList (if the actorList contains me, whether the result of the (the actorList).getOne (me) is 1 or 0). If it is not it is inserted in the list. As soon as it is put in the actorList, the message stepFrame is generated every 1/60 of a second. It is that pulsation that is made use of when the timer is actually set to work. Once the stepFrame is generated by the object, the rest is not so complicated. The timer determines the difference between a given time stored in a variable and the excess amount of seconds (the number of seconds programmed to wait) and subtracts the former from the latter. Once the condition expressed by the comparative operator is satisfied, the parent script executes the lines of commands to repopulate the text member with the next text until all the string values are exhibited. To add one more surprise element to the presentation, the location of the text member is varied each time a new string value is inserted into it. Therefore, if the coding is used as it is presented here, the viewer is not only challenged with the new text every so many seconds but also with the visual exertion he has to force himself to in order to recognize the textual content in the first place. If we add the sound accompaniment whatever the new elements are presented that will make the intellectual exercise more vigorous and render the whole presentation aesthetically more engaging.

Works Referenced

Abrams, M.H., ed. *The Norton Anthology of English Literature.* New York: W.W. Norton, 1993.

Allenson, Andrew, et al. *Director 8.5 Studio.* Birmingham, UK: friends of ED, 2001.

Armstrong, Jay, et al. *Macromedia Director 8.5 Shockwave Studio.* San Francisco, CA: Macromedia, 2001.

Barthes, Roland. *S/Z.* New York: Hill and Wang, 1986.

Catanese, Paul. *Director's Third Dimension.* Indiana: QUE, 2001.

Culler, Jonathan. *Structuralist Poetics.* New York: Cornell University Press, 1982.

Harbage, Alfred, ed. *William Shakespeare: The Complete Works.* New York: The Viking Press, 1975.

Jakobson, Roman. *Language in Literature.* Cambridge, Massachusetts: Harvard University Press, 1987.

Rosenzweig, Gary. *Special Edition: Using Macromedia Director 8.5.* Indiana: QUE, 2002.

文学的テキストとその視覚的表現の可能性について

文学的作品を伝統的なメディアから解き放ちそれを「読者」が異なる視点から享受するにはどのような可能性があるのだろうか。そのような前提のもとにこの論文では多元的レベルでの経験主義的なアプローチを中心としたインターアクティブ性の強い文学作品の提示の仕方を模索している。具体的には「読者」とのインターフェイスの最大限の活用を目指すためのスクリプティングの開発を織り交ぜながら文学作品の真髄をマルチメディアを駆使したマルタイパースペクティブな手法で伝達する可能性を追求している。