Dynamic Graphics and the Low Bandwidth Communication Barrier

John Amanatides

University of Toronto

ABSTRACT

Graphics systems that use a low bandwidth communication line to transmit images to a display processor suffer from slow updates when images of moderate complexity are to be displayed. This is particularly evident when animation is attempted. But there is often no need to transmit an entire frame. Dynamic variables can be used to reduce the communication traffic from the host by transmitting only the differences between frames. The animation capabilities of the display processor can thus increase dramatically. Also, dynamic variables provide a method of simplifying applications programs. These ideas are extended to non-refresh graphics systems such as Telidon.

RESUME

Les systemes graphiques qui utilisent des lignes de communication a faible capacite pour transmettre des images a un processor d'affichage souffrent de la lenteur a rafraichir les images d'une certaine complexite. Ce devient particulierement evident dans le cas de l'animation. Mais il n'est souvent pas necessaire de transmettre l'image entiere. Les "variables dynamiques" definies ici peuvent etre utilisees pour reduire le volume des communications de l'ordinateur hote en ne transmettant que les differences entre images. La capacite du processeur d'affichage a transmettre les images s'accroit alors considerablement. De plus, les variables dynamiques offrent le moyen de simplifier l'ecriture des programs d'application. Les memes idees peuvent etre etendues aux systemes sans rafraichissement tel que Telidon.

1. Introduction

A bottleneck facing many graphics systems is the low bandwidth communication line connecting the display processor to the host. Typically 9600 baud, this is too slow to transmit complete images at anything approaching 30 frames a second. At this rate, for example, only thirty three characters can be transmitted in real time. Modifying existing images is also a problem. The traditional solution to the update problem has been to divide the display file into segments. Now, only a portion of the display file need be retransmitted. But even this approach is unsatisfactory if real time animation[x] is attempted. There is simply too much data to be transferred. To increase the speed at which animation can occur we must find a way of reducing further the data that must be forwarded by the host.

2. An Observation

If we look carefully at the display file during an animation sequence we see that there is very strong frame to frame coherence. Though there may be differences between frames it is usually the same image, slightly modified, that is displayed. For example, if we are stretching a triangle by moving a vertex, the number of lines and the general shape will remain constant. The only difference between frames is that the vertices of the triangle are at different positions on the screen. If we look at the display file describing these frames we see that the same series of instructions are included. Only the arguments to the graphic primitives describing the image are different.

What is needed, therefore, is a faster way to modify the arguments to these primitives. By only modifying the arguments, we can hope to reduce the amount of data transfer. The problem that immediately arises is how to perform this display file modification in a structured manner that also removes unnecessary cognitive burden from the programmer. Any sort of pointer manipulation by the programmer could quickly become unwieldy as segments are constantly created and destroyed.

Suppose we introduce a new data type called *Dynamic*. This data type has the property that if a variable is reassigned then all graphic primitives that have used that variable are modified to reflect its new value. Its properties are illustrated in the example below:

```
Dynamic a;
..
..
a=1;
CreateSegment(1);
MoveAbs2(0,0);
LineAbs2(0,1);
LineAbs2(1,a);
LineAbs2(0,0);
CloseSegment();
..
..
a=10;
```

In the above example we draw a triangle using calls to CORE-like functions. But the result of executing the last statement changes the position of the third vertex from (1,1) to (1,10). Thus, modifying the triangle only requires reassigning the dynamic variable a. To animate the triangle by having a vertex move is as simple as executing the C statement *for(a=0;a < 50; a++);*.

This approach is conceptually simple. Changes to the screen can now easily and conveniently be made. There is a simple one-to-one correspondence between changing the dynamic variables and modifying the image on the screen. But more importantly, dynamic variables provide the mechanism with which we can reduce the communication requirements of animation by sending to the graphics processor only the dynamic variable's new value.

3. DPAC

DPAC is a graphics package written at the University of Toronto's Computer Systems Research Group that encorporates dynamic variables [AMAN 83]. The host computer is a 11/45 with the graphics processor being SPIWRIT [BAEC 81, MILL 81]. SPIWRIT contains a bit-splice processor with a 16k display file and two 256 by 256 by 4 bit frame buffers. Every thirtieth of a second, SPIWRIT decodes the display file and writes the resulting image into the double buffered frame buffer

DPAC generates the familiar segmented display file. Dynamic variables are implemented with the help of a preprocessor. It converts assignments and expressions containing them into function calls that set and get the value of the dynamic variable. To the user, dynamic variables are a data type identical to integer except that they have the property of modifying the display file.

Implementing dynamic variables on SPIWRIT involved changing the microcode that decoded the graphic primitives. Graphic primitives are now allowed to have arguments that are addressed indirectly. A portion of the display file is used to store these variables.

[x] Througout, the term animation is intended in the general sense, that is, any dynamic behavior on the screen that can occur in real time.

There are some disadvantages associated with implementing dynamic variables. Because DPAC dynamically alters the display file certain simplifications that are normally made by most graphics packages must be abandoned. There can be no display file optimization for SPIWRIT. For example, multiple **MoveRel** cannot be collapsed into one **MoveRel** because they may be later modified by dynamic variables. Some graphics processors do not have **LineAbs** primitives. They are not required because a **LineAbs** can be simulated with a **LineRel** from the previous point. This is no longer true with DPAC since that intermediate point can be affected by a dynamic variable.

4. Modification Primitives

Dynamic variables have given us the mechanism to reduce the communication needs of transmitting animation sequences. But we can also use them to provide the basis for more powerful graphics primitives to further reduce the amount of data that must be transmitted. For example, let us introduce the **Interpolate** primitive. It takes as arguments a dynamic variable, a starting and ending value and a period of time. It's function is to interpolate the dynamic variable between the two end points during the specified time. The earlier animation sequence consisting of the moving vertex can now be encoded into this primitive. More importantly, the communication requirements for this sequence reduce to zero as this primitive can be executed locally by the graphics processor. By using other dynamic variables for the endpoints and time, complex sequences can be specified.

Interpolate is an instance of a whole class of "modification" primitives that can be used to reduce data transfer. Segments containing these primitives can contain different "scripts" of animation sequences with the appropriate script being "performed" by the graphics processor whenever the related segment is posted. (A more detailed discussion of the concept of scripts and actors is found in [REYN 82].) Further research is required to generate a reasonable set of these primitives.

5. Hardware Enhancements

We have described how to transmit animation sequences in real time. But to fully utilize them we must be able to scan out the display file in real time also. We now discuss how to build raster graphics systems with inexpensive hardware to perform the animation in the required time.

5.1. Clipping

Traditionally, a display processor has not had to perform clipping. Objects that are to be displayed but are off the screen are clipped by the graphics package on the host during display file compilation. Thus, no part of the display file has instructions to draw off the screen. The tracking symbol however, may in fact be partially off the screen and a simple form of clipping called scissoring is used. The display processor still draws the tracking symbol but the underlying circuitry detects an attempt to write to a portion of memory off the screen and does not allow it to occur. The time wasted by this unnecessary drawing is small and thus there is little motivation for including true clipping circuits.

DPAC unfortunately can have significant portions of the display file off the screen. These objects cannot be removed from the display file because later dynamic variable modifications can bring these hidden objects back into view. Simple scissoring will now waste too much time. Clipping is required. At present this is done in software by the microprocessor for every primitive. This process slows down display file scan-out so that fewer objects can be displayed in real time.

We suggest an alternative scheme to remove this burden from the microprocessor. The circuitry that detects when scissoring is to be performed can be modified to interrupt the microprocessor whenever an off-the-screen write is attempted. In this way the processor does not have to waste time writing off the screen or trying to clip visible objects. This is especially important when drawing polygons. Judicious use of this interrupt can also speed up the clipping process for objects that are known to be on the border of the screen.

5.2. Multiple Viewports

Having multiple viewports on the screen is useful but is usually implemented in software on the host machine. We propose a scheme where this can be accomplished on the graphics processor with no increase in scan-out time. This scheme is an extension of the clipping proposal mentioned above. Instead of raising an interrupt when the processor attempts to write off the screen the interrupt is now raised when a write is attempted outside the current viewport. This can easily be accomplished by adding four hardware registers and comparators. These registers would contain the extent of the viewport. Additional graphic primitives to set these variables would be implemented. It would then be possible to modify the viewport with dynamic variables. This feature could be used to create novel real time animation.

5.3. Parallel Writes

The biggest bottleneck with most real time graphics processors writing into a frame buffer is the speed at which the processor can write to the frame buffer. For example, if we have a 256 by 256 frame buffer and are capable of writing to a pixel in 500 nanoseconds, it would take one thirtieth of a second to simply set the background colour! To build a graphics processor with a high resolution frame buffer and still expect to have a reasonable number of

polygons drawn in real time is out of the question. There must be a faster way to write into the frame buffer.

A way out of this impasse is to implement frame buffer memory circuitry similar to that proposed by Whelan [WHEL 82]. Briefly, his approach is to design a slightly modified memory chip to accomplish very fast frame buffer writes. Instead of having inputs to specify a unique address the chip inputs four numbers: the position and size of a rectangle in the frame buffer. In one memory write cycle it sets the pixels in this rectangle to the indicated colour. To implement polygon filling with this circuitry the microprocessor would go down the left and right edges of the polygon, setting a whole scan line in one write cycle. Thus, the scanning out of polygons can occur at over an order of magnitude faster than at present.

The only drawback to this approach is that the pixels written in the above manner are all set to the same colour. Accordingly, smooth shading of polygons is not possible. Slight modifications to the memory chip, however, would allow simple textures to be drawn.

6. Future Applications

A good area in which dynamic variables can be applied is in Telidon. Telidon is the Canadian videotext system in which graphic primitives known as Picture Description Instructions (PDI) are sent down low speed communication lines to a decoder in the home [OBRI 82]. PDIs were chosen because they are structured, resolution independent and are very compact. Telidon suffers from slow update times when images of moderate complexity are to be displayed because of the low bandwidth communication lines. Dynamic variables can be used to reduce the traffic from the central site by transmitting only the differences between frames. Also, dynamic variables provide a structured environment in which animation can easily and effectively be expressed. The bandwidth requirements for simple animation can be further reduced if modification primitives are used. Now the host need not modify the dynamic variables directly but instead instructs the PDI decoder to perform this locally.

Since Telidon is not a refresh system and does not have a local display file the above techniques pose a problem. But even a small display file in the decoder would be sufficient to store the animation commands. Frame buffer animation techniques [BOOT 82] can then be used to display the resulting animation sequence in real time. Whole animation sequences can be stored and later "performed" when necessary without loading down the communication line. New implementations of Telidon decoders could encorporate the hardware enhancements outlined above to provide the capability for real time animation.

7. Concluding Remarks

In conclusion, by using dynamic variables we can maximize the bandwidth utilization of the communication line to the graphics processor. Static portions of the display file are no longer sent to the graphics processor repeatedly. Only the differences between frames are retransmitted. Modification primitives can further reduce transmission by modifying dynamic variables locally.

8. Bibliography

[AMAN 83] J. Amanatides, "DPAC: A Dynamic Graphics Package for a Real-Time Raster Device", Master's Thesis, Department of Computer Science, University of Toronto, Jan. 1983.

[BAEC 81] R. Baecker, D. Miller and W. Reeves, "Towards a Labratory Instrument for Motion Analysis", Computer Graphics Vol 15(3): pp 191-197, July 1981.

[BERG 77] S. Bergman and A. Kaufman, "Association of Graphic Images and Dynamic Attributes", Computer Graphics Vol 11(2):pp. 18-23, Summer 1977.

[BOOT 82] K.S. Booth and S.A. MacKay, "Techniques for Frame Buffer Animation", Proceedings Graphic Interface '82, pp. 213-220, May 1982.

[CSUR 75] C.A. Csuri, "Computer Animation", Proc. SIGGRAPH '75, pp. 92-101, June 1975.

[FOUR 81] A. Fournier, "A Proposal for a Four-dimensional Graphics System" Proceedings of the Seventh Canadian Man-Computer Communications Conference, pp. 371-375, June 1981.

[MILL 81] D.H. Miller, "A Two-Dimensional Dynamic Display System", Technical Note CSRG-24, Computer Systems Research Group, University of Toronto, August 1981.

[OBRI 82] C.D. O'Brian et. al., "Telidon- Videotext Presentation Level Protocol: Augmented Picture Description Instructions", Communications Research Centre Technical Note No. 709-E, Ottawa Canada, Feb. 1982.

[PFIS 76] G.F. Pfister, "A High Level Language Extension for Creating and Controlling Pictures", Computer Graphics Vol 10(1):pp. 1-9, Spring 1976.

[REEV 76] W.T. Reeves, "A Device-Independent, General-Purpose Graphics System in a Minicomputer Time-Sharing Environment", Master's Thesis, Department of Computer Science, University of Toronto, January 1976.

[REEV 81] W.T. Reeves, "Inbetweening for Computer Animation Utilizing Moving Point Constraints", Computer Graphics Vol 15(3): pp 263-269, July 1981.

[REYN 82] C.W. Reynolds, "Computer Animation with Scripts and Actors", Computer Graphics Vol. 16(3): pp. 289-296, July 1982.

[WHEL 82] D.S. Whelan, "A Rectangular Area Filling Display System Architecture", Computer Graphics Vol 16(3): pp. 147-153, July 1982.