The Graphics Environment for Max

Mark
Danks
University of California, San Diego
mdanks@ucsd.edu

Abstract The Graphics Environment for Max, or GEM, is a real-time computer graphics language which operates under the SGI version of Max. GEM supports abstract computer graphics as well as image and pixel manipulations, enabling the user to control the graphics as objects, instead of as a sequence of vertices. Because it maintains the Max paradigm, GEM is an easy to use development environment.

1 Introduction

The recent port of Max [IRCAM, 1995], from the IRCAM Signal Processing Workstation (ISPW) board on the NeXT workstation to the Silicon Graphics (SGI) platform, allows the extension of the Max environment to the realm of computer graphics. Since SGI has shifted its computers from the proprietary graphics package GL to the platform independent OpenGL, a graphics environment based on OpenGL will be compatible with almost any computer platform on which Max is developed. The Graphics Environment for Max (GEM) is a collection of external objects that uses OpenGL and runs under the SGI version of Max.

1.1 Goals

Since a general environment for real-time graphics was being developed, it was important to have a set of criteria for each of the design decisions. GEM was developed to:

- Follow the Max paradigm
- Be user friendly
- Be easy to extend
- Be efficient enough to run in real-time

MIDI and digital audio systems [Puckette, 1991] operate along with graphics and video systems in GEM, rather than in two competing environments. GEM was designed so that the user can create any 3-D object and manage it as *one object,* not a series of coordinates in 3-D space. A number of different graphics packages are based on this paradigm, such as [Howard, *et al.,* 1992], however, they were created solely for graphical user interface design.

A logical control flow starts with the object which creates the shape, followed by objects that manipulate the graphic, like `rotate,` and ending with the `render` object to display the image. A collection of commands that creates a graphic is called a `gemList.` Whenever a `render` object receives a `gemList,` it executes all of the commands contained within it. Figure 1 shows a simple patch to display and rotate a red square.

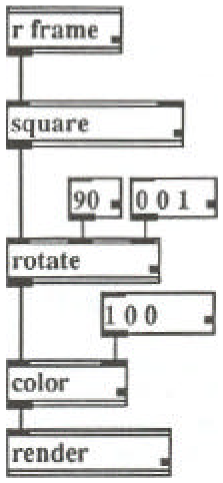

Figure 1: Simple GEM patch

1.2 OpenGL

OpenGL is a state based system. This means that there is only one state, consisting of the various rotations, translations, and scaling that have been applied, that is always active in an OpenGL context. When a program calls an OpenGL function that actually draws pixels on the screen, the location of each individual pixel is calculated by the current state. Every rotation, translation, and scaling can be represented by a 4x4 transformation matrix (x y z w). By calculating the matrix for a specific manipulation and then multiplying it by the current matrix, a new 4x4 matrix is generated, representing the current state.

As the various rotations and translations are performed, the current matrix is modified by the appropriate transformation matrix. The matrix does not modify the actual vertices in the graphic. When the time comes to display the graphic, each vertex is simply multiplied by the current transformation matrix, generating the actual pixel to display on the computer screen. As explained later, the *manipulators* do not have to "know" anything about the graphics which they are modifying because they only change the transformation matrix. A much more detailed explanation of state based systems and transformation matrices appears in Introduction to Computer Graphics [Foley *el al.,* 1994].

2 Usage

GEM consists of five types of objects: *geometrics (geos)*, *non-geometrics (non-geos)*, *pixes*, *manipulators*, and *controls*. The *geos* are objects which create a concrete shape. The *non-geos* consist of objects which display something on the screen, but do not have a defined shape. The pix objects perform pixel operations and generate images. The *manipulators* control and transform the various graphics. The *controls* have global effects and manage the graphics window.

2.1 Geometrics

The *geometrics*, or *geos*, are simple forms that give the user a group of objects which can be used as building blocks for more complex shapes. The current *geos* are square, circle, triangle, cube, sphere, cone, polygon, and curve. Every time a *geo* receives a bang, it generates the graphic and sends a gemList out of its outlet. This graphic can then be modified by any of the *manipulators* to suit the user's needs.

2.2 Non-geometrics

The *non-geometrics*, or *non-geos*, are the objects which display something on the screen, yet do not have a defined shape as cube does. Thus far, the only *non-geos* are light and world_light. GEM supports up to eight lights, the typical limit for OpenGL platforms (although it is system dependent). The color of the light can be set with a list of three numbers, indicating RGB values, and lights can also be rotated and translated. light is local to the scene while world-light is considered to be at an infinite distance away (all of its rays are parallel). light produces more realistic highlights and can be positioned within the scene, however it takes much more processing power.

2.3 Pixes

The pix objects are all of the GEM externals which deal with image displaying and processing. The following pix objects exist: image, pix_buf (pixel buffer), pix_render, pix_gain, pix_mask, pix_blend, pix_2grey, pix_convolve, pix_threshold, and pix_composite. *Pixes* attempt to minimize the amount of processing power that is needed, especially when the same effect is being used repeatedly. Pix_mask and pix_composite often work together to blend together images. One simple example of this is to simulate blue-screen compositing. Edge detection, blurring, and filtering can all be accomplished with pix_convolve, depending on the matrix used in the convolution.

2.4 Manipulators

The *manipulators* provide control over the graphics through rotate, translate, color, and merge. The rotate and translate objects control the transformation matrix which applies to each GEM object. For both *manipulators*, the user gives a rotation or translation vector and then a value. The rotation value controls the angle of rotation in degrees, while the amount of translation is dependent on the window viewpoint. The affects on the transformation matrix are cumulative and order dependent

The merge object allows the user to join two gemLists together. As some users have pointed out, it does not actually merge the g em L i s r s together, but instead creates one graphic out of the original two graphics so that all subsequent operations by the manipulators apply to the combined, new object. This also resets the axes of the object to the equivalent of 0, 0, 0. By constructing a complex shape with the building blocks that the *geos* provide, the user can encapsulate the collection in a Max patch, in effect creating a new *geo*.

2.5 Controls

The *controls* have a global affect on the graphics window. These objects include gemwin, camera, and render. The graphics window manager is gemwin. It modifies the global state variables for OpenGL such as lighting, optimization, creation and destruction of the graphics window, and window size. gemwin is GEM's interface to the X window system and is responsible for accessing the X display, render is the object which actually displays the graphic on the computer screen. Whenever render receives a gemLiSt, it executes all of the commands contained within it, such as rotate or color, and displays the resulting graphic. The camera is the user's viewpoint in the graphic window and can be modified as any other object. It can be rotated and translated, creating a feeling of motion as the view moves throughout the graphical space.

2.6 Realities

A number of tools and utilities were created during GEM's debut in the production of *headingsouth* [Danks, 1995a], a composition for solo `cello, real-time sound processing, and real-time computer graphics. One of the biggest challenges was controlling the frame rate and generating the frame bang. Because objects like light need to be manipulated before the graphics are rendered, one send object was not sufficient. The solution was for gemwin to send a bang out when it had switched the frame buffer. The blending pix objects only work correctly when the graphics are rendered from back to front. This dilemma prompted the frame abstraction which sends a series of bangs, each delayed by some proportion of the frame rate. While the user still has to correctly coordinate the drawing order, GEM provides the triggering. Other techniques were designed to produce visual delays, creating a trailing effect. Simple effects, such as rotating objects with speeds that were multiples of each other, produced graphics reminiscent of John Whitney with his graphical resonances and harmonics [Whitney, 1980].

3 Implementation

One of the key elements of GEM is its ability to utilize OpenGL. The following sections describe the technical implementation of GEM.

3.1 Display Lists

OpenGL display lists are the key to GEM. Display lists are collections of OpenGL commands which can be executed with one function call. For example, a display list could encapsulate all of the drawing commands to create a sphere (potentially a very high number of commands depending on the resolution), which could then be called at any time by the program. Each display list is assigned a reference number, which GEM stores inside the `gemList`. Since OpenGL is a state based system, all transformations that have been executed apply to the display list. The OpenGL Programming Guide [Neider *et al.*, 1993] describes `glcallList()` as follows: "The commands in the display list are executed in the order they were saved, just as if they were issued without using a display list" (p. 128).

While there is some processing overhead in creating the initial display list, the execution of display lists is not any slower than executing the commands contained within them [Neider *et al.*, 1993]. It is much faster to simply call the display list of the graphic rather than recompiling the object every time it will be used. There is some overhead involved in jumping to a display list, but if there are any matrix muttiplications, computations, or images, the speed gained is substantial. Since the display list only contains computed values, and not variables, objects must recompile their display lists if any of the variables describing the shape are modified. When the display lists are created, they do not have to render the graphics immediately. The *geos* compile the display list, but `render` executes them.

Because the display list is referenced by an integer, it is trivial for Max to pass the list number between objects. Since the objects are not defined in Max by a series of vertex lists or line segments, but instead by one number, a Max list message is used to transmit the display list. This also removes a global structure which might contain information about the various shapes. OpenGL maintains the display lists so that the only communication between objects that is necessary is the display list number. To create a large memory structure for shapes would not only be inefficient, but also extremely difficult in Max using only externals. The `rotate` object, for example, does not manipulate the actual object shape, but merely appends `r angle x y z` to the `gemList` [Danks, 1995b]. Except for the object which actually creates the list, objects do not have any information about the display list that they are modifying.

3.2 OpenGL Options

OpenGL is a powerful graphics environment, providing more options than the average user needs. Currently, many of the options take a large amount of computer processing, curtailing GEM's ability to function in real-time. Taking advantage of the flexibility of OpenGL, GEM can turn the various options on and off, including global adjustment such as `optimize`, which disables as many OpenGL options as possible. OpenGL options include lighting, smooth blending, fog, and other effects. Since the various options are global within a GLX context, the `gemwin` object is able to set each state at the window creation time. The states that GEM has access to are limited to those for which it has been programmed. However, if a user wants to turn on a feature which the basic GEM objects do not use, it is easy to create a new object which will do so.

3.3 Pix Objects

Just as a `start` message must be sent to the `dac~` to enable signal processing, a `pix` message must be sent to all of the pix objects to activate the image processing. This creates a pix-block, a topology of the objects which modify a specific image. `image` reads in a RGB file and establishes the pix-block. If each pix maintains its own copy of the image, then the processing is reduced since objects "up-stream" from a pix which has been changed do not have to recompute the image; however the memory requirement is extremely high. If no objects store the image, then the memory required is low, but each object must re-compute the image every frame. As a compromise, `image` and `pix` but monitor the pix objects and only send out a new image when an object sets a modified flag. If the user always wants the image processed with `pix_2grey`, then the image is only processed the first time and stored in the `pix_buf`. Figure 2 demonstrates the use of a `pix_buf` with `pix_threshold`.

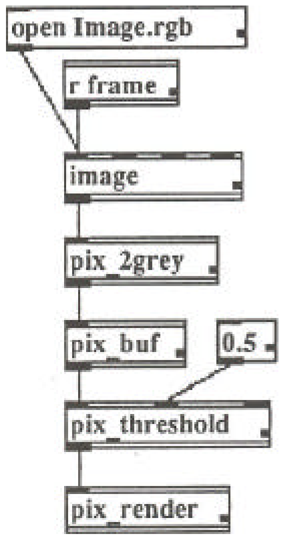

Figure 2: Manipulating an image

3.4 Creating New Objects

A major goal of GEM is to provide an interface for real-time computer graphics that parallels the Max data control paradigm. In some areas, this has limited the power of GEM, but only if the user limits himself to the use of existing objects. As in Max, one of the true powers is creating externals. The development framework for GEM makes it easy to create new objects; usually, only one function must be modified. Even an object as potentially complex as sphere needs little work to make it function smoothly. When the sphere object receives a bang, it checks to see if the graphics window is different from when its display list was first created and if an internal modification flag has been set. If the display list must be rebuilt, only one function is called.

4 Future Developments

GEM will become more effective on desktop workstations as computer power increases in the coming years. Real-time video and more extensive image processing are prime targets, especially implementing beyond the current objects. The video library which SGI includes with its computers provides easy access to the video data stream. Preliminary test programs indicate that it will take more computer power than a basic Indy can provide. However, just as real-time audio DSP has become possible in software, so too will real-time video DSP.

One area that needs more work is the problem of increasing user flexibility in the underlying design of GEM. Three problem areas which stand out are: 1) how to deal with objects with a very large number of points; 2) how to group objects together coherently; 3) how to create objects the user can define. One solution is the creation of a "little language" where users can define an object that a GEM object could parse.

5 Summary

GEM is an easy to use environment for real-time graphics built on top of the Max signal processing program for the SGI platform. Unlike the Macintosh version of Max, users are able to manipulate entire objects instead of vertices and points in order to define unified shapes. The environment is accessible to many different levels of users, from novices who just want to put simple graphics on the screen, to experts who are writing their own externals that work directly at the OpenGL level. By combining the audio processing ability of Max with the power of OpenGL, GEM is a system which has much artistic potential.

References

[Danks, 1995a] Danks, Mark. headingsouth (San Diego, CA: 1995).

[Danks, 1995b] Danks, Mark. *GEM* Developer Notes, http://man104nfs.ucsd.edu/-mdanks/GEM/ GemDev.html (San Diego, CA: 1995).

[Foley et al., 1994] Foley, James, Andires van Dam, Steven Feiner, John Hughes, and Richard Phillips. Introduction to Computer Graphics (Reading, Mass.: Addison-Wesley, 1994).

[Howard et al., 1992] Howard, Robert, and Frank Zinghini. "ObjectGraphics," Computer Graphics Using Object-Oriented Programming (New York: John Wiley and Sons, 1993).

[IRCAM, 1995] IRCAM. IRCAM Software, http: //www.ircam.fr/produi ts/logici els/Iste-logiciels-e.html#max (Paris, France: IRCAM, 1995).

[Neider et al., 1993] Neider, Jackie, Tom Davis, and Mason Woo. OpenGL Programming Guide (Reading, Mass.: Addison Wesley, 1993).

[Puckette, 1991] Puckette, Miller. "Combing Event and Signal Processing in the MAX Graphical Programming Environment," Computer Music Journal, 1513, (Cambridge, Maxx.: MIT Press, 1991), 68-77.

[Whitney, 1980] Whitney, John. Digital Harmony; on the complementarily of music and visual gq (Peterborough, NH: Byte Books, 1980).