Introduction of DIPS Programming Technique

Chikashi Miyama[1], Takayuki Rai[1], Shu Matsuda[2], Daichi Ando[3]
[1]Sonology Department, Kunitachi College of Music
[2]Digital Art Creation
[3]Chalmers University of Technology
email: rai@kcm-sd.ac.jp and shu@dacreation.com

Abstract

DIPS, (Digital Image Processing with Sound), is a set of Max objects that handle real-time visual image processing events and OpenGL functions in the jMax GUI programming environment. In this paper we introduce DIPS programming techniques and programming strategies for interactive multimedia art. DIPS for Linux and Mac OS X has been released under GPL and can be downloaded from http://dacreation.com/dips.html.

1. Introduction

DIPS consists of more than a hundred OpenGL functions, various video effect objects, QuickTime objects, 3D model handling objects, movie handling objects, image file handling objects, etc. Since it was developed as a tool for using OpenGL in a flexible real-time environment, knowledge of OpenGL is essential. However, just as Max offers complex programming opportunities for people who do not necessarily have programming skills, DIPS was also designed for composers and creators with an aim to exploiting graphical calculations and realizing interaction between sound and visual events.

2. DIPS programming

Most DIPS objects are bang oriented, meaning that calculations are executed only when an object receives a bang in its left-most inlet. In this way we can control the rendering interval and avoid unnecessary calculations, thus saving machine-processing power, since slower machines can be used with a slower bang-rate (i.e. metro speed). A second type of DIPS object does not require any bang. For instance, the DIPSWindow object creates a rendering window right after the object is instantiated.

A) Initialization procedure

We first create a window where various 3D objects and images will be rendered. Normally the following initialization procedure consisting of six steps is used to create the window. (Fig.1).

1) DIPSWindow : With this object we can specify name, size in pixels, and the position of the window on the desktop. For a TV-sized monitor display the size of 640 by 480 is appropriate. In example-1 we create a 320 by 240 pixel window at the position 50 and 50 (x, y from top-left position of the desktop) with the name "sampleWindow".

2) DIPSSetCurrentWindow : With this object we specify the current target window.

3) DGLClearColor : This specifies clear values (RGB each range [0, 1]) for the color buffer. In this case, we specify the background color of the window. This process can be executed together with the object "DGLClear".

4) DGLMatrixMode : This determines the current matrix. Here we set it to "GL_PROJECTION" matrix stack temporarily.

5) DGLLoadIdentity : This sets the current matrix with an initial value (identity matrix).

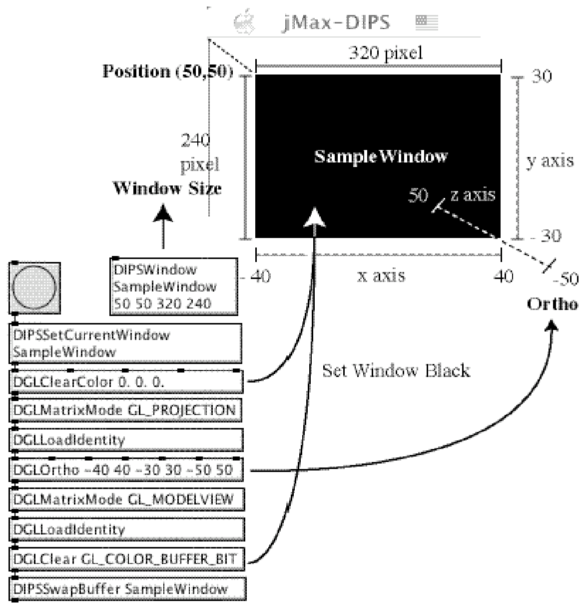

Fig.1 : example-1

6) DGLOrtho : This produces a parallel projection, the orthographical 3D space that is defined with values: left/right on the x-axis, bottom/top on the y-axis, and near/far on the z-axis. These values set the scale of three-dimensional space.

A perspective projection can be produced using the "DGLFrustum" or "DGLUPerspective" objects. (Fig.2).

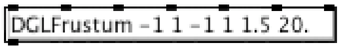

Fig.2 : DGLFrustrum

7) DGLMatrixMode : We set the matrix to "GL_MODELVIEW".
8) DGLLoadIdenttity : The current matrix is set with an initial value.
9) DGLClear : This clears the specified buffer. In this case we clear the color buffer ("GL_COLOR_BUFFER_BIT") and set new values defined by the previous "DGLClearColor".
10) DIPSSwapBuffer : Finally the result of the above process is rendered in the specified window using a swap buffer function.

In most situations, DIPS objects are programmed between the last "DGLMatrixMode" and "DIPSSwapBuffer". Here we created just an empty black window. Also in this example the object "DGLViewPort" is omitted. It defines the rendering rectangular or square region in the window in the case where the user wishes to render in an area smaller than the entire window.

B) Drawing a simple polygon

Usually 3D models in computer graphics consist of a large number of vertices, and each group of a few vertices forms a polygon surface. Thus 3D models consist of a large number of polygon surfaces. In the following example we draw a simple triangular surface in our "sampleWindow". (Fig.3). All objects introduced in this example must be placed between the "DGLClear" and "DGLSwapBuffer" objects.
1) DGLColor : This object simply defines the color (RGB) of the object or vertex. In this example, the color parameters are set to [0. 0. 1.], so that a blue triangular surface is rendered in the window. To create several different colored objects we have to specify the color value with "DGLColor" before each target DIPS object.
2) DGLBegin and DGLEnd : Here with the argument "GL_POLYGON" we declare a polygon with vertices. (In "DGLBegin" we can specify "GL_LINES_STRIP" to draw a line, or "GL_QUADS" to draw a quadrangle, etc.) To close this instruction we must place a "DGLEnd" object at the end of the sequence.

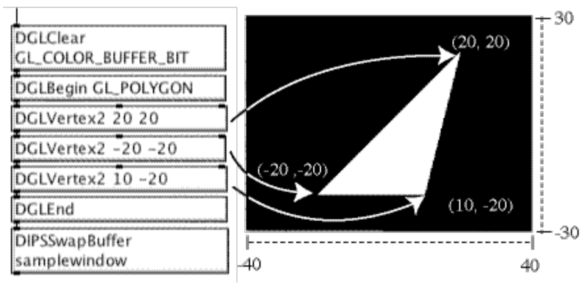

Fig.3 : example-2

3) DGLVertex2 and DGLVertex3 : There are two DIPS GL objects to specify vertices. "DGLVertex3" defines vertices in three dimensions (x, y, z), and "DGLVertex2" does in two dimensions (x, y) where the z-axis value is automatically set to zero.

C) Creating a 3D model with the DGLUT object

We can also create a 3D model with DIPS GLUT objects, which are wrapper objects of the OpenGL Utility Toolkit. These DGLUT objects allow us to create simple geometric models with just one instruction. Geometric models that can be created with DGLUT objects include Cone, Cube, Dodecahedron, Icosahedron, Octahedron, Sphere, Teapot, Tetrahedron and Torus. They can be rendered in Wire or Solid form. (Fig.4).

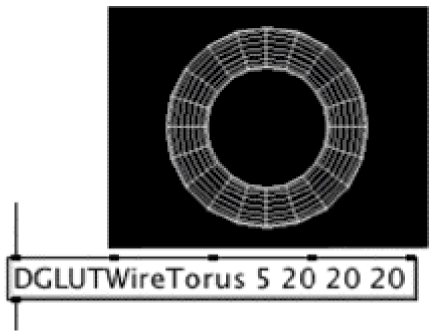

Fig.4 : DGLUTWireTorus

D) Translating a model object

To translate, to change the size, and to rotate an object, we use the following three DIPS GL objects.
1) DGLTranslate : This object moves the position of the model on the x, y, and z-axis. The values of parameters for this object depend on the scale value that is defined in the "DGLOrtho", "DGLFrustum", or "DGLUPerspective" object beforehand.

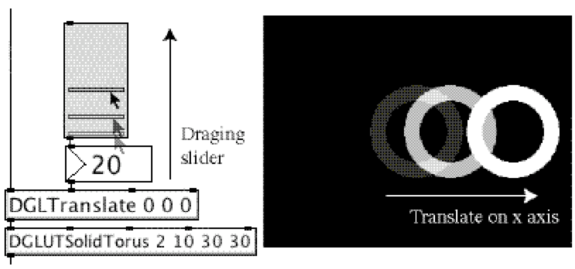

Fig.5 : example-3

2) DGLRotate : For rotating the model we use this DGL object. The last three arguments, x, y, and z coordinates of a vector, determine the axis for the rotation. The first argument specifies the angle in degrees to rotate. For example, when we wish to rotate the model 30 degrees on the y-axis we set the object's arguments to the following: "DGLRotate 30 0. 1. 0.".

3) DGLScale : This object reduces and extends the size of the model on the x, y, and z-axis.

These three objects are usually placed just before the specification of a target model DIPS object, such as "DGLBegin", "DGLUT", etc. In Fig.5, a jMax number box and a jMax slider object are connected to the second inlet (x-axis) of a "DGLTranslate" object. By dragging the slider value with the mouse you can translate the position of the Torus, specified by "DGLUTSolidTorus", on the x-axis.

To translate several models individually, in most of cases we recommend using "DGLPushMatrix" and "DGLPopMatrix" objects. Each target model DIPS object must be placed in between these two objects. A "DGLPushMatrix" object pushes the current matrix value and keeps it in the matrix stack. Conversely, "DGLPopMatrix" throws away the current matrix value and goes back one before in the matrix stack.

E) Setting light and material

We can specify lighting effects and features of surface material of the model.

1) DGLEnable and DGLDisable : Before we specify lighting effects, we have to enable OpenGL lighting calculations using the "DGLEnable" object. With this and the "DGLDisable" object we can turn on and off various OpenGL calculations. In this example (Fig.6), first we enable "GL_LIGHTING" calculations, and then we turn on "GL_LIGHT0" (the "number zero light"). The number of lights we can use depends on the hardware environment, but we have found that one can use at least eight lights in a minimal environment.

2) DGLLight : With this object we can chose one of ten different sorts of lights and also define its color (RGBA=Alpha). The first argument specifies the light to use, in the example, "GL_LIGHT0".

3) DGLMaterial : This object defines the quality of materials on the face of the model. We can specify one of four material modes: "GL_AMBIENT", "GL_DIFFUSE", "GL_SPECULAR", "GL_EMISSION".

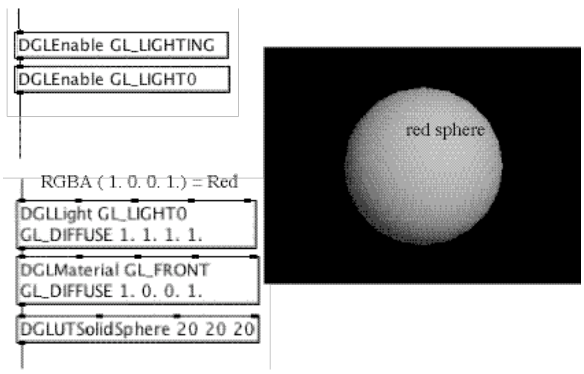

Fig.6 : example-4

We can also decide its RGBA parameters for each mode. How the face of the model looks is the result of the combination of specified light and material. In the example, the material is set to red and the light is blue, making the model appear purple. (Fig. 6).

F) Texture mapping

Texture mapping is a well-known and useful technique in computer graphics that allows artificial objects to appear more complex and more vivid. We can glue a two dimensional image to a polygon (Fig.7). Texture mapping calculations can be enabled when "DGLEnable GL_TEXTURE_2D" is executed.

Fig.7: texture mapped rectangles

1) DIPSPixTable : This object loads various kinds of movie and image files which QuickTime support. First, we specify the name of a pixel table, then the width and height of a movie or image file, the number of frames in the case of movie file, and the search-path or URL of the source file. (Images and movies can be loaded via the Internet.) The width and height of the texture must be a power of two. The source image will be resized according to the specified texture size.

2) DGLTexImage2D : This determines which "DGLPixTable" will be attached to the polygon surface as a texture.

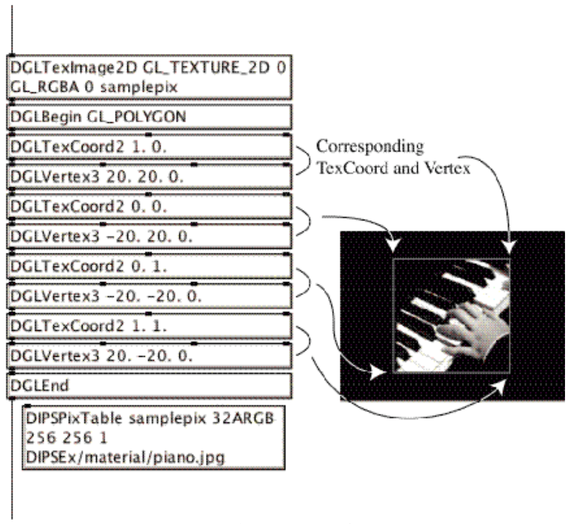

Fig.8 : example-5

3) DGLTexCoord2 : A position (x and y, each with range [0, 1]) in the image is attached on a vertex of the model defined by a "DGLVertex3" object. (Fig.8).

If we are working with a movie file or stored sequential images, we must specify the current image with its frame number using the “DIPSMakeCurrentTable” object. Its first argument is the name of a DIPSPix-Table, and the second is the frame number. Frame numbers can be continuously changed at the right inlet of the “DIPSMakeCurrentTable” object. Thus, we can play the movie at any speed, forward or backward, etc.

G) DIPSVideoIn :

This object handles video input images from a digital video camera, so that incoming video images can be brought into a DIPS patch. (Fig.9). It refreshes or captures video input when it receives a bang. The input can be transformed with various DIPS pixel calculation (DPX) objects, it can be analyzed, or it can be used as moving texture. (Fig.10). Currently only a single Firewire (IEEE1394) camera input is supported.

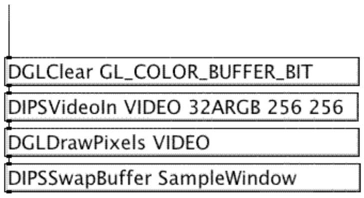

Fig.9 : DIPSVideoIn

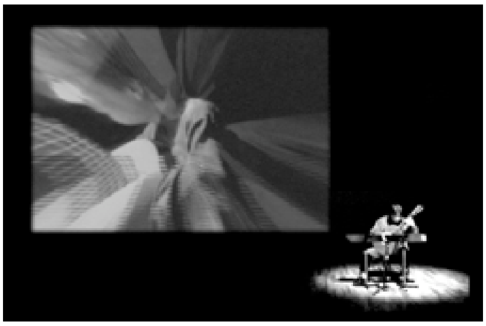

Fig.10: in-coming video signal used as a video texture

3. Conclusion

In this paper we have introduced basic DIPS programming techniques. The DIPS objects we have briefly described are fundamental objects. There are many more DIPS objects available, along with online help. The DIPS distribution includes a large number of DIPS patch examples (Fig.11), and tutorial patches. In addition we include supporting tools for programmers interested in writing DIPS external objects.

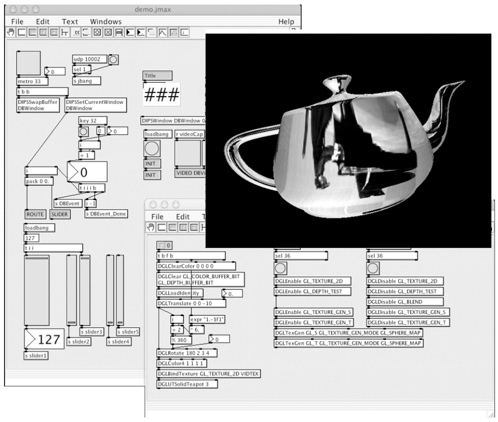

Fig.11 : DIPS patch example

Future plans include porting DIPS to the new jMax4 environment, while continuing to enhance pixel calculation and motion detection objects.

4. Acknowledgements

We would like to thank the real-time group at IR-CAM for their support.

Reference

[1] Matsuda, S., Miyama, C., Ando, D., Sakai, Y., “DIPS2: a multimedia programming environment on jMax”, 2003-MUS-51, 2003.
[2] Matsuda, S., Miyama, C., Ando, D., “DIPS for Linux and Mac OS X”, 2002-MUS-48, 2002.
[3] Matsuda, S., Miyama, C., Ando, D., Rai, T., "DIPS for Linux and Mac OS X”, in Proceedings of the International Computer Music Conference 2002.
[4] Matsuda, S., Rai, T., “DIPS: the real-time digital image processing objects for Max environment”, in Proceedings of the International Computer Music Conference 2000.
[5] Woo, M., Neider, J., Davis, T., Shreiner, D., OpenGL Architecture Review Board (1999). OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2(3rd ed.). Addison-Wesley.
[6] OpenGL Architecture Review Board, editor: Kempf, R., Frazier, C., (1999). OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.1(2nd ed.). Addison-Wesley.