HD (in) Processing

Andrés Colubri
Design|Media Arts dept., UCLA
Broad Art Center, Suite 2275
Los Angeles, CA. 90095-1456
+1 310 825 9007
acolubri@ucla.edu

Abstract

In this paper I describe the new GLGraphics library for the Processing programming language and environment. This library integrates Graphics Processing Unit(GPU)-accelerated effects and textures into the Processing API (Application Program Interface). GLGraphics solves the current limitations of Processing when handling High-Definition video and images in real-time, while maintaining the ease and familiarity of the existing API. This library makes programming with standard shading languages (such as GLSL and Cg) more accessible to the visual artist/designer who uses Processing for creating computer-based artworks. GLGraphics also brings to Processing techniques of GPGPU (General Processing on GPU), which can substantially speed-up simulations of particle systems and other types of complex real-time effects. The usefulness of this library becomes apparent when considering today's intense work in the contexts of interactive installations, augmented reality, data visualization, computer vision, etc., where the need for real-time processing of large amounts of data has become a constant necessity. GLGraphics is undergoing active development at this time, but it already has a significant level of usability and performance. It is an open-source project, released under the GPL. The source code, as well as binary packages, documentation and tutorials can be accessed from this website: http://users.design.ucla.edu/~acolubri/processing/glgraphics/home/index.html

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object oriented programming.

D.1.5 [**Computer Graphics**]: Graphics Utilities.

J.5 [**Arts and Humanities**]: Arts, fine and performing.

General Terms

Algorithms, Documentation, Computer Graphics, Digital Arts

Keywords

Processing, OpenGL, shader, texture, GLSL, Cg, library, effect, real-time, HD, GPU, video, filter

1.Introduction

Processing [1] is a widely used language designed as a learning environment as well as a production tool for digital artists. It was originally created by Casey Reas and Ben Fry in 2001 at the Aesthetics and Computation Group at the MIT Media Lab. One reason for its popularity is the simplicity of the programming environment and the convenient syntax of the language.

On the one hand, with the increased popularity of interactive installations, techniques of augmented reality, computer vision, utilization of HD content, etc., the need for real-time processing of large amounts of data has become a constant necessity in the digital arts.

On the other hand, the rapid development of GPUs is turning them into very powerful parallel processors which can be used not only to render very complex graphics in real-time, but also to speed-up the types of computations mentioned in the previous paragraph. Since the functionalities of modern programmable GPUs are accessible through the standard graphics library OpenGL [2], we have built GLGraphics on top of OpenGL in order to encapsulate some of these functionalities and to make them easily available in Processing.

Summarizing, the GLGraphics library has the three following goals:

1) To integrate OpenGL textures and GPU texture effects into the Processing API by means of its library mechanism.

2) To allow the generation of real-time effects on high-resolution media such as HD videos and large images.

3) To bring closer to visual artists using Processing the techniques of GPGPU, which turn the graphics chip into a parallel co-processor able to handle very quickly certain types of complex simulations and effects.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACE'08, December 3–5, 2008, Yokohama, Japan.
Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

2. Description of the library

GLGraphics consists in a collection of classes that wraps-up different OpenGL features, such as textures, GPU shaders, Frame Buffer Objects (FBO) and Vertex Buffer Objects (VBO), etc. The main two classes of the library are GLTexture and GLTextureFilter. GLGraphics also includes a new Processing renderer that enables off-screen drawing, among other features.

GLTexture is a descendant of the core class PImage. Hence, the use of GLTexture combines seamlessly with the already existing image-handling functionality, while adding new methods to manipulate OpenGL textures.

GLTextureFilter encapsulates GPU-accelerated texture effects called "filters". Examples of these filters are Gaussian blur and edge detection, High Dynamic Range (HDR) tone mapping, etc., and also more complex GPU effects like the simulation of particle systems, which can be expressed as a combination of multiple texture filters chained together.

2.1OpenGL textures

The GLTexture class brings support for OpenGL textures to the core PImage class. The texture can be thought as a new layer added on top of the pixels property of PImage. An image is copied from the canvas of the PImage to the pixels array by calling the loadPixels() function. Similarly, the pixel data can be transferred to the texture by calling loadTexture(). In a reverse order, the updateTexture() copies the texture data to the pixels property, while updatePixels() puts the pixels into the drawing canvas.

However, transferring texture data from the GPU memory to the pixels array on CPU memory in order to draw the texture on the screen is a slow operation, specially for large textures. In order to avoid this bottleneck, the GLGraphics library includes a new renderer which draws OpenGL textures directly without requiring any GPU-CPU transfer. The GLGraphics renderer can also be used as an off-screen drawing surface. This allows to create multiple independent canvases inside the same Processing sketch.

A GLTexture can be initialized in different ways, as shown in the following code snippets:

```
// Loading an image directly upon creation:

tex = new GLTexture(this, "image.jpg");

// Creating a texture of a given size:

tex = new GLTexture(this, 200, 200);

// Created empty, sized to a given resolution, and then

// using the pixels array to put data in it:

tex = new GLTexture(this);

tex.init(100, 50);

tex.loadPixels();

for (int i = 0; i < 5000; i++) tex.pixels[i] = 0xff000000;

tex.loadTexture();
```

2.2Texture filters

A texture filter, encapsulated in the GLTextureFilter class, is basically a GPU shader that operates on an input texture or group of textures and writes the output of the rendering operation to another texture(s). The shader can be imagined as a computational kernel that is executed on the GPU, such as a Gaussian blur or emboss effect [3-6]. One advantage of this approach is that the calculation is actually off-loaded to the GPU, freeing CPU resources for other operations, such as handling user interaction, general flow of the program, etc.

The configuration of a filter is saved in a xml file, where the names of the shaders that define the filter are stored. An entire shader program can consist in a vertex, geometry and fragment shaders, corresponding to each one of the programmable stages of modern GPUs. With the latest release GLGraphics at the time of writing (0.8), the shaders have to coded in the OpenGL Shading Language (GLSL or GLslang) [3]. Future releases of the library will include support for the Cg shading language from Nvidia [4].

A texture filter works following the Processing API for the built-in image filters, and the following is a sample program that applies a simple blur filter on a image loaded from the disk:

```
import processing.opengl.*;

import codeanticode.glgraphics.*;

size(200, 200, GLConstants.GLGRAPHICS);

// Initializing source and destination textures:

GLTexture texSrc = new GLTexture(this, "image.jpg");

GLTexture texDest = new GLTexture(this,

                    texSrc.width, texSrc.height);

// Initializing filter:

GLTextureFilter blur = new GLTextureFilter(this,"blur.xml");

// Applying the filter on texture texSrc,

// and writing the result to texDest:

texSrc.filter(blur, texDest);

// Drawing the resulting image:

image(texDest, 0, 0, width, height);
```

In this particular example, the xml file just lists the filename of the fragment shader that contains the Gaussian kernel, some basic description strings and the number of input and output textures supported by the filter, here one in both cases:

```
<filter name="gaussian blur">

  <description>3x3 Gaussian blur kernel</description>

  <fragment>blur.glsl</fragment>

  <textures input="1" output="1"></textures>

</filter>
```

The shader blur.glsl contains the GLSL code for a standard Gaussian blur convolution kernel, as shown here:

```
uniform sampler2D src_tex_unit0;
uniform vec2 src_tex_offset0;
void main(void) {
 float dx = src_tex_offset0.s;
 float dy = src_tex_offset0.t;
 vec2 st = gl_TexCoord[0].st;
 // Apply 3x3 gaussian filter
 vec4 color = 4.0 * texture2D(src_tex_unit0, st);
 color += 2.0 * texture2D(src_tex_unit0,st + vec2(+dx,0.0));
 color += 2.0 * texture2D(src_tex_unit0,st + vec2(-dx,0.0));
 color += 2.0 * texture2D(src_tex_unit0,st + vec2(0.0,+dy));
 color += 2.0 * texture2D(src_tex_unit0,st + vec2(0.0,-dy));
 color += texture2D(src_tex_unit0,st + vec2(+dx,+dy));
 color += texture2D(src_tex_unit0,st + vec2(-dx,+dy));
 color += texture2D(src_tex_unit0,st + vec2(-dx,-dy));
 color += texture2D(src_tex_unit0,st + vec2(+dx,-dy));
 gl_FragColor = color / 16.0;
}
```

The uniform variables src_tex_unit0 and src_tex_offset0 in the shader code above represent the first texture unit and the offset of that texture. These names are a convention that has to be necessarily followed by the GLSL shaders used in the GLGraphics filters, otherwise the library cannot send the texture data to the shader (i.e.: a second input texture unit has to be named src_tex_unit1, and so on).

2.3 Floating-point textures

The GLGraphics library also adds support for floating-point textures in Processing. These textures can store floating-point values in each one of its four (RGBA = XYZW) components. Another important concept in the context of GPGPU is that of ping-pong textures [5, 6]. Since a texture could be only read or write, but not both simultaneously, an operation that needs to update a texture based on its previous values requires in fact two textures that are swapped continuously. This is, after the step where one of the textures has been used as input (read) and the other as output (write), the role of the textures are exchanged, so the one where the latest data was written to is now used as input. This technique is called ping-pong, and there is a class in GLGraphics called GLTexturePingPong which facilitates this operation.

2.4 GPGPU: particle systems

Simulations of large particle systems constitute a type of computation well suited for GPGPU acceleration [5, 6], given its inherently parallel nature. The binary distribution of the library comes with a couple of particle systems as examples. The "Simple GPU Particle System" program implements a system where the velocities of the particles are controlled by the position and motion of the mouse. This example contains two filters used as computational/rendering kernels: one to simulate the motion of the particles, and another to draw the particles on the screen. See "Figure 1" for a typical visual output of this example.

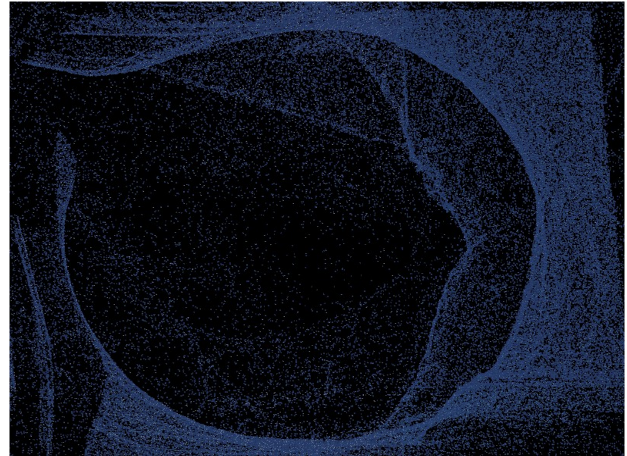

Figure 1. Output of the SimpleGPUParticleSystem running **more than 100,000 textured particles in real-time**

More sophisticated particle systems can be implemented, as exemplified by the "Painter" program also included with the library (see "Figure 2"). In this algorithm, the gradient of the luminance of the input image is averaged iteratively to generate a vector field with many vortexes and flows [7]. The motion of the particles follow this vector field. Since all the computations are performed on the GPU, the effect can be applied on real-time video.

Figure 2. Painterly-rendering algorithm implemented within the GLGraphics framework. It uses a large particle system to create the appearance of flowing paint in real-time.

2.5 HD playback

One particular weakness in Processing is the playback of high-resolution video files, since the core video library is based on Quicktime. There is no good integration between Quicktime and Java, the underlying language on which Processing is based, and this results in poor framerates when playing large videos.

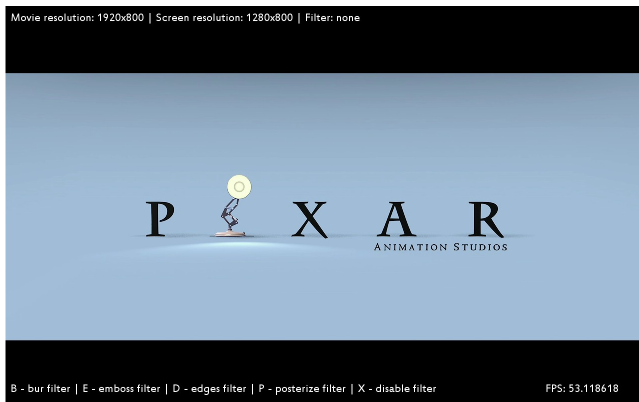

Figure 3. Processing playing an 1080i HD clip (1920x800 pixels) at 30 fps in fullscreen mode, using the GLGraphics and GSVideo libraries.

The open source multimedia framework GStreamer [8] works better in combination with Processing than Quicktime, and this makes possible to play high resolution video files through the GSVideo library [9].

Figure 4. Processing playing an 1080i HD clip and applying a GPU posterize filter on it.

GSVideo re-implements the API of the core video library with GStreamer-java [10], and the playback performance results to be substantially better. However, to obtain acceptable framerates with HD movies, the rendering of the video frames has to be accelerated through OpenGL (see "Figure 3"), using GLGraphics.

The method consists in reading the frames with the GSMovie object, and then copying the movie pixels into a GLTexture object which is then rendered to the screen, as shown in the following sample program:

```
import processing.opengl.*;

import codeanticode.gsvideo.*;
import codeanticode.glgraphics.*;

GSMovie movie;
GLTexture tex;
void setup() {
  size(640, 480, GLConstants.GLGRAPHICS);

  movie = new GSMovie(this, "hdclip.mov");
  movie.loop();

  tex = new GLTexture(this);
}
void movieEvent(GSMovie movie) {
  movie.read();
}
void draw() {
  background(0);
  if ((1 < movie.width) && (1 < movie.height)) {
    // Copy the latest movie frame into the texture tex:
    tex.putPixelsIntoTexture(movie);
    image(tex, 0, 0, width, height);
  }
}
```

Once the video frame is copied inside a GLTexture, further real-time post-processing is possible with GPU filters. For example, "Figure 4" shows the result of applying a posterize filter on a 1080i HD clip. With a relatively low-end graphics card, such as a Nvidia Geforce 8400, the framerate remains above 25 fps.

3.Conclusions

Initial testing with the GLGraphics library has been highly positive, since it shows that it is possible to apply real-time filters on high-resolution images and video, while following Processing's language and usage conventions.

Large particle systems can be successfully simulated within the GLGraphics framework, with sizes up to 1,000,000 particles.

Since this library is an extension of pre-existing core functionality, this opens-up the possibility of combining previous techniques with the GPU-based approaches to create new and original effects.

Future development of this library will add further options such as Cg support and better hardware compatibility across as many OpenGL2-compatible video cards as possible.

4.Acknowledgments

The author would like to thank Casey Reas and Ben Fry for their comments and suggestions, and the Processing community of users and developers for their constant feedback and encouragement.

5.On-line resources

The homepage of the GLGraphics library is the following:

http://users.design.ucla.edu/~acolubri/processing/glgraphics/home/index.html

In this site there is full access to the entire documentation of the library, as well as usage examples. Source code and binary packages are available at the sourceforge page of the project:

http://sourceforge.net/projects/glgraphics/

6.References

[1] Reas, C.E.B, Fry, F. 2007. Processing: A Programming Handbook for Visual Designers and Artists. The MIT Press.

[2] Shreiner, D. 2006. The OpenGL Programming Guide Version 2.0. Addison-Wesley Professional.

[3] Rost, R. J. 2005. OpenGL Shading Language. Addison-Wesley Professional.

[4] Randima, F., Kilgard, M. 2003. The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Addison-Wesley Professional .

[5] Pharr, M. 2005. GPU Gems 2: Programming Techniques for High performance Graphics and General-Purpose Computation. Addison-Wesley Professional.

[6] Nguyen, H. 2007. GPU Gems III. Addison-Wesley Professional.

[7] Wang, C. M., Lee, J. S. 2004. Non-Photorealistic Rendering for Aesthetic Virtual Environments. J. of Information Science and Engineering. Vol. 20, Num. 5 (Sept. 2004), 923-948.

[8] GStreamer. Open source multimedia framework: http://gstreamer.freedesktop.org/

[9] GSVideo. GStreamer video library for Processing: http://users.design.ucla.edu/~acolubri/processing/gsvideo/home/

[10] GStreamer-java. Java interface to the gstreamer framework: http://code.google.com/p/gstreamer-java/