Radar Display GPU Coding With The Graphics API

Prita Sharma[1], Venkata Nagabhushanam[2], Rekha Bhandarkar[3], L.Ramakrishnan[2] , Ravi Prakash Reddy M[2]

1 System Engineer Trainee, Infosys Technologies, Mysore, India
2 Central Research Laboratory, Bharat Electronics Limited, Bangalore, India
3 Dept. of ECE, Nitte Mahalinga Adyanthaya Memorial Institute of Technology, Nitte, India
pritasharma25@gmail.com

Abstract: The graphics display plays a crucial role in a Radar System. The main objective is to achieve a configurable software based scan converter for radar display with rich graphic features on an indigenous embedded hardware. An accurate, effective and real-time radar display is built by coding a graphics GPU, which is coded with help of OpenGL API.

Keywords- radar display, GPU, Embedded, OpenGL, graphics hardware

I. Introduction

RADAR scan conversion, which is needed to transform polar coordinate ultrasound data into Cartesian coordinate data consumable by standard graphics systems, is one of the performance bottlenecks of a Radar Display system. Many current systems employ custom FPGA based hardware or DSPs to create scan converters. However with the increasing speed of computers and their graphics systems, PCs are now a viable platform for Radar Display systems. This paper proposes using the texture mapping capabilities of OpenGL and leverage the graphics system to perform scan conversion module of Radar Display. The new OpenGL algorithm effectively off-loads the heavy lifting of scan conversion, the thousands of interpolations that must be performed, to the GPU. This should dramatically improve the throughput of the conversion and free up CPU cycles on the general purpose processor for other tasks. This paper demonstrates the viability of using texture mapping feature of OpenGL to perform scan conversion.

With the advancement of Embedded Computing Graphics Processing technologies, radar displays are changing in terms of performance and rich user interface presentation. Modern Graphical Processing Unit [3] (GPU's) is sophisticated processors offering multiple cores and pipelines. GPUs support their own graphics language to run. Figure 1 shows Modern day computer that has a dedicated Graphics Processing Unit (GPU) to produce images for the display, with its own graphics memory (or Video RAM or VRAM).The GPU technology has found a niche in embedded systems are variants of desktop or laptop graphics cards, featuring GPU's, onboard texture memory. This architecture is combined with optimized Open-GL[3] drivers for high-speed color bits enables software-based video keying at very high speeds. This means that graphics comprising underlay and overlays can be seamlessly fused with radar, or other sensor video, to create a composite display that even surpasses the display capabilities of some dedicated hardware L[9] radar scan converters. OpenGL is popular in the embedded graphics community and hence being the choice for this implementation. It offers the most appropriate feature set, also is well supported by the COTS graphic chip providers. OpenGL implementation provides application developers a flexible way to produce graphics [4] applications.

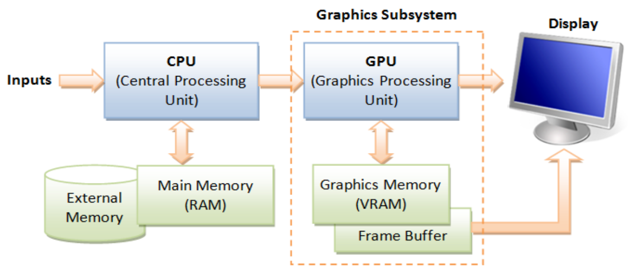

Figure 1. Computer Graphics Hardware

II. Basics Of Embedded Graphics

I APIs to code GPUs

OpenGL is a cross platform library that is used for interfacing with programmable GPU's for rendering 3D graphics. OpenGL[14] is a standard that has grown to be a large API[7]. Standardized subsets hence provide a key to using OpenGL in safety-critical and deep embedded environments, eliminating redundant capability and stressing simplicity and small footprint.

In computer graphics, rendering is process of producing image on the display screen from the given modular description. This OpenGL simplistically is used in our application to draw visuals. Figure 2 shows the rendering pipeline of OpenGL, which provides the approach to draw 2D and 3D data with help of primitive's elements[10] (points, lines, triangles, line strips, triangle strips, etc). The elements are specified using a sequence of vertices, with each vertex containing multiple data (position, color, surface normal, texture coordinates).

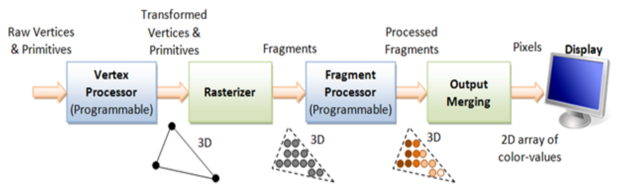
Raw Vertices & Primitives
Transformed Vertices & Primitives
Fragments
Processed Fragments
Pixels
Display
Vertex Processor (Programmable)
Rasterizer
Fragment Processor (Programmable)
Output Merging
3D
3D
3D
2D array of color-values

Figure 2. The OpenGL Rendering Pipeline

The rendering pipeline has four stages mainly:

- Transform and process individual vertices.
- Convert each connected vertices (primitive) into fragments. A fragment is a pixel which is aligned with the pixel grid with attributes (like color, position, normal, texture).
- Process the individual fragments.
- The fragments of all primitives are combined into 2D color-pixel for the output display.

The GPU's can also be coded with GLSL[11]. That is the vertex and fragment processing stages are programmable known as vertex shader and fragment shader. The rasterization and output merging stages on the other hand are not programmable but are configurable (via commands issued to the GPU).

II Radar Scan Conversion

The radar-scan[5] data received is the range and azimuth information that is in polar coordinates system and in order to be displayed on a 2-Dimentional display screen it needs to be converted to Cartesian format and this conversion is known as Radar scan conversion[15]. As the radar antenna moves around the azimuth increases steadily, a number of pulses are transmitted to detect targets around the site. Typically can be 4096 pulses (12 bit encoding), 16384 pulses (14 bit encoding) or 65536 (16 bit encoding) in serial form per 360 degree of rotation are known as the ACP (Azimuth Count Pulses). The pulses assumed in this software code implemented are 4096; however the implementation is flexible to be altered to desired pulses. After every complete rotation of the antenna new scan data is generated, which is converted and hence updated on display screen. The main task is to convert these data to Cartesian (x, y) display coordinates, creating the necessary display pixels. As the input data available to display system is polar coordinate form, when displayed as it is will produce a B-mode frame. Thus scan conversion is needed to transform the polar coordinate data into standard Cartesian coordinates that are needed by standard graphics subsystems. Figures 3 and 4 illustrate a B-mode frame of a radar scan data before and after scan conversion.

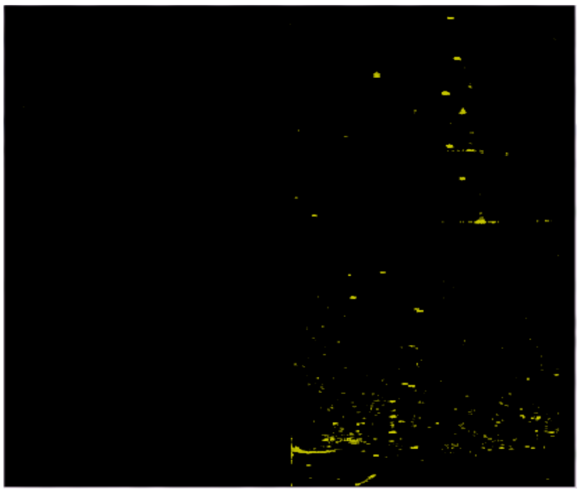

Figure 3. Raw B-mode RADAR data

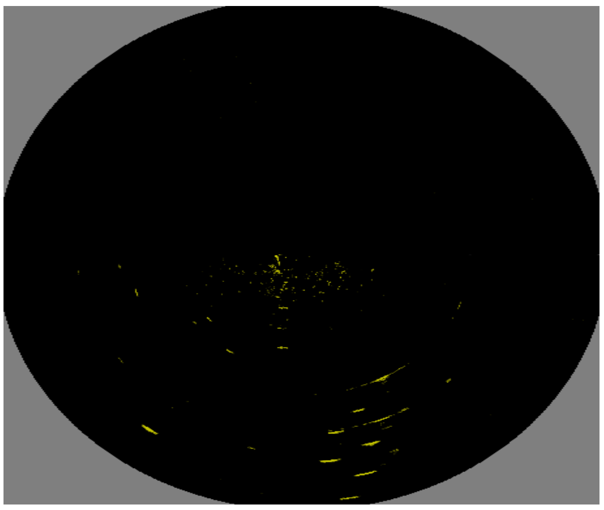

Figure 4. Scan-converted radar data.

III. IMPLEMENTATION

There are a number of transformation methods to perform radar scan conversion; however the most efficient in our perspective is used. This implementation uses the texture mapping function of OpenGL to map the raw B-mode data onto a sector shape that fits the dimensions of the scan converted image. The sector is defined using adjacent quadrilaterals (quads) as shown in Fig. 5. Two of the vertices of each quad are always at the origin (also the axis of rotation in the mechanical system) and the other two vertices lie on the outside of the sector at distance r from the origin. As the quads are defined, texture coordinates are mapped to each vertex. Each texture coordinate refers to a specific location in the raw B-mode data, and once a texture coordinate has been assigned to all four vertices of a quad the OpenGL library can map that portion of the data onto the sector. The code excerpt in Fig. 6 demonstrates how the quads and texture coordinates are defined. Each time through the loop four vertices are defined at the specified angle, two at the origin and two at the edge of the sector. The texture coordinates mapped to these vertices correspond to the beginning and ending points of the vectors in the B-mode image.

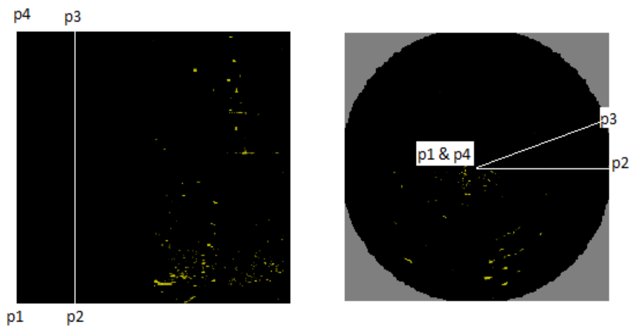

Figure 5. Texture vertices mapped to Quadrilateral vertices.

```
for(int i=0;i<sectors;i=i+1)
{
  glBegin(GL_POLYGON);
          glTexCoord2f(0.0,(float)i/(float)sectors);
          glVertex2f(0.0,0.0);
          glTexCoord2f(1.0,(float)i/(float)sectors);
          glVertex2f(cos((double)i*2*M_PI/sectors) * r,
          sin((double)i*2*M_PI/sectors) * r);
          glTexCoord2f(1.0,(float)(i+1)/(float)sectors);
          glVertex2f(cos((double)(i+1)*2*M_PI/sectors) * r,
          sin((double)(i+1)*2*M_PI/sectors) * r);
          glTexCoord2f(0.0,(float)(i+1)/(float)sectors);
          glVertex2f(0.0,0.0);
  glEnd();
}
```

Figure 6. Code illustrating sector definition and texture coordinate mapping.

Once the entire mapping is complete, the OpenGL library performs all of the necessary interpolations to define each pixel in the final image. OpenGL can be directed to handle the interpolations in two basic ways: nearest and linear. Using the nearest option, the library finds the data point in the texture (the B-mode data) and uses its value for the value of the pixel, just like the nearest neighbor algorithm. Similarly the linear option directs the library to perform bilinear interpolation using the four closest data points in the texture. In this implementation the linear option (bilinear interpolation) is used. Since the interpolation occurs within the OpenGL library this implementation can leverage the processing power of the graphics hardware of the PC. Furthermore, OpenGL is platform independent and can interact with the graphics hardware of any standard PC. These advantages should provide better throughput for the radar display system. This method of implementation has taken on average 0.02 seconds per frame.

CONCLUSION

A GPU accelerated software based scan converter for radar display with rich graphic features is achieved using Texture mapping feature of OpenGL library. The software code implementation for the radar display for 4096 ACPs was successfully written using the above stated algorithm. It was run and tested on visual studio 2008 express version, with help of default radar data input files. In terms of throughput the new OpenGL algorithm is much faster than the traditional linear interpolation and bilinear interpolation algorithms. By leveraging the power of the graphics system of the PC the OpenGL algorithm is not only able to provide more throughput, it off-loads most of the work from the general purpose CPU. With the performance bottleneck of scan conversion pushed onto the GPU, the CPU is now free to perform other operations necessary for the Radar Display System such as handling user input, or data management. Using OpenGL is not only a relatively easy way to implement a radar scan converter, it is a much more efficient use of a systems resources.

ACKNOWLEDGMENT

The authors express sincere acknowledgements to Mr. Mahesh.V Chief Scientist, CRL, BEL, Bangalore and Management of Bharat Electronics Limited for encouragement and support to write this paper. Also authors express sincere thanks to Mr.P.Ramesh Babu Senior Research Staff and Mr.Ranganath Member Research Staff CRL, BEL for their helpful contribution in this research work.

REFERENCES

[1] Prita Sharma, Venkata Nagabhushanam, Rekha Bhandarkar, L.Ramkrishnan,“Multi layered Graphics engine using high speed Embedded platform for RADAR applications”, ICETE 2013.

[2] George W.Stimson, “*Introduction to AIRBORNE RADAR*”, 2nd edition SciTECH Publishing,Inc.

[3] Aaftab Munshi, Dan Ginsburg, Dave Shreiner,“*OpenGL®ES 2.0 Programming Guide*”, Addison-Wesley publication.

[4] Richard S.Wright,Jr., Benjamin Lipchak, Nicholas Haemel, “*OpenGL Super Bible*”,4th edition, Addison-Wesley Publication.

[5] Norman lin, “*Advanced linux 3d graphics programming*”, 2001 Wordware Publishing Inc.

[6] Brett Borden, “Radar Imaging of Airborne Targets”, IOP Publishing Ltd 1999.

[7] Phil Cole, “*OPENGL ES SC – OPEN STANDARD EMBEDDED GRAPHICS API FOR SAFETY CRITICAL APPLICATIONS*”, 24th Digital Avionics Systems Conference October 30, 2005.IEEE,VOL 2.

[8] Mark Snyder, Quantum3D, Glendale, AZ, “Solving the embedded OpenGL puzzle- making standards, tools and API's work together in highly embedded and safety critical environments” 24th Digital Avionics Systems Conference October 30, 2005.IEEE,VOL 2.

[9] Niklas Peinecke, Hans-Ullrich Doehler, and Bernd R. Korn, “*Simulation of Imaging Radar Using Graphics Hardware Acceleration*” , 2008 SPIE Digital Library Proc. of SPIE Vol. 6957 69570L-1.

[10] Huang Xiying, Shao Wei, XU Renji, Lining, "*Modeling and Application of Three-Dimentional Special Data Field for Radar Detection Range*",2012 International Conference on Computer Science and Electronics Engineering, 978-0-7695-4647-6/12.

[11] OpenGL Common/Common-Lite Profile Specification, V 1.0.02, The Knronos Group, 2004.

[12] OpenGL ES Safety Critical Profile Specifications, V 1.0. the khronos Group, 2005.

[13] JungHyun Han, "*3D Graphics for Game Programming*", CRCPress, 2011.

[14] OpenGL mother site @ www.opengl.org.

[15] Radartutorial (www.radartutorial.eu)

[16] Online lecture:http://cs.berkeley.edu/~ravir

[17] Nate Robin's OpenGL Tutor @http://www.xmission.com/~nate/opengl.html.

[18] Nehe OpenGL Tutorials @ http://nehe.gamedev.net.

[19] Scan conversion @http://www.cambridgepixel.com

BIO DATA OF AUTHOR(S)

Miss Prita Sharma is currently undergoing training at Infosys Technologies as a System Engineer Trainee at Mysore. She had completed her M.Tech in VLSI Design and Embedded Systems from NITTE in 2013 and B.Tech (ECE) from RIT in 2011. Was an intern at CRL-BEL Bangalore for over 10 Months during 2012-13 as a part of her project phase in her M.Tech. She is a specialist in embedded systems and has specialised in C++ and Java from NIIT.

Ms. Rekha Bhandarkar completed her B.E in ECE in 1989 and M.Tech. in Digital Electronics in 2001. She has submitted her thesis for Ph.D. on "Cochlear filter implementation on FPGA". She is currently an Associate Professor in the Dept. of Electronics and Communication at NMAM Institute of Technology Nitte. Her areas of interest include Signal Processing,VLSI and Embedded Systems.

Mr. J Venkata Nagabhushana J is a B.Tech (CSE) & ME (CSE). He started his career in embedded systems for RADAR applications and his areas of interest include Design & Development of Linux and Vx Works Board Support Packages for Power PC SBCs, Implementation of Multi-target Tracking algorithms and RADAR Display Software. Currently he is a Member SRS at CRL, BEL.He is involved in design and development of embedded computing products & signal processing systems for radar applications. He has been bestowed with BEL & R&D Excellence award.

Mr.L.Ramakrishnan obtained B.Tech (Electronics Engg) MIT, Anna University & ME (ECE) OU. Starting his career at HAL, worked in projects including Air borne Transponders, Airborne radar for modern Indian fighter aircrafts, Air Route Surveillance Radar, Radio Proximity fuses etc. His area of professional interest includes Design and Development of front ends for Radar, Wireless & Communications Systems. He is currently serving as Member (SRS) & Group Head at Central Research Laboratory, BEL. He presently is involved in the design and development of embedded computing products & signal processing sub systems for radar applications. He has over twelve reputed publications to his credit. As part of team, he has been bestowed with Raksha Mantri's award under Innovation Category and BEL R&D Excellence Awards. He is Member (IEEE) and certified PMP.

Mr. Ravi Prakash Reddy M is a B.Tech (CSE) & M-Tech(CSE) from NIT Calicut. He is Working as Member Research Staff in Central Research Laboratory, BEL. He fields of intrest are of Radar display Systems, VxWorks and Linux Board Support packages for PowerPC SBC.