iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 1 **Issue:** III **Month of publication:** October 2013

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

Vol. 1 Issue III, October 2013
ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Performance Analysis of Sobel Edge Detection Filter on GPU using CUDA & OpenGL

Ms. Khyati Shah

Assistant Professor, Computer Engineering Department
VIER-kotambi, INDIA khyati30@gmail.com

Abstract: *CUDA(Compute Unified Device Architecture) is a novel technology of general-purpose computing on the GPU, which makes users develop general GPU (Graphics Processing Unit)programs easily . GPUs are emerging as platform of choice for Parallel High Performance Computing. GPUs are good at data intensive parallel processing with availability of software development platforms such as CUDA (developed by Nvidia for its Geforce series GPUs).Basic goal of CUDA is to help pro- grammars focus on the task of parallelization of the algorithms rather than spending time on their implementation. It supports the Heterogeneous computation where applications use both the CPU and GPU. In this paper we propose the implementation of sobel edge detection filter on GeForce GT 130 on MAC OS using CUDA and OpenGL .We reduces the Global Memory using kernel function. Also compare their results and performance to the previous implementation and it gives the more optimized results.*

Keywords: *CUDA, GPU, Image Processing, OpenGL, PBO, Sobel, VBO*

INTRODUCTION

CUDA (Compute Unified Device Architecture) is a new architecture for issuing and managing computations on the GPU [1]. Nvidia GPU is available for GeForce Series, Quadra Series and Tesla brands. CUDA architecture supports a range of computational interfaces including OpenCL (Open Computing Language) and DirectX Compute. CUDA brings the C-like development environment to programmers for the first time, which uses a C compiler to compile programs, and replaces the shader languages with C language and some CUDA extended libraries. Users needn't map programs into graphics APIs anymore, so GPGPU program development becomes more flexible and efficient. More than one hundred processors resided in CUDA graphics card schedules hundreds of threads to run concurrently, resolving complex computing problems [2].

OpenGL is a software interface to graphics hardware. It is designed as a hardware-independent interface to be used for many different hardware platforms. OpenGL uses the prefix gl for core OpenGL commands and glu for commands in OpenGL Utility Library. Similarly, OpenGL constants begin with GL and use all capital letters. OpenGL also uses suffix to specify the number of arguments and data type passed to a OpenGL call.

CUDA and OpenGL by utilizing a PBO (Pixel Buffer Object) to create images with CUDA on a pixel-by-pixel basis and display them using OpenGL. CUDA to generate 3D meshes and utilize OpenGL VBOs (Vertex Buffer Objects) to efficiently render meshes as a colored surface, wire frame image or set of 3D points. To focus on CUDA rather than OpenGL, we have used an OpenGL framework that can mix CUDA with both pixel and vertex buffer objects.

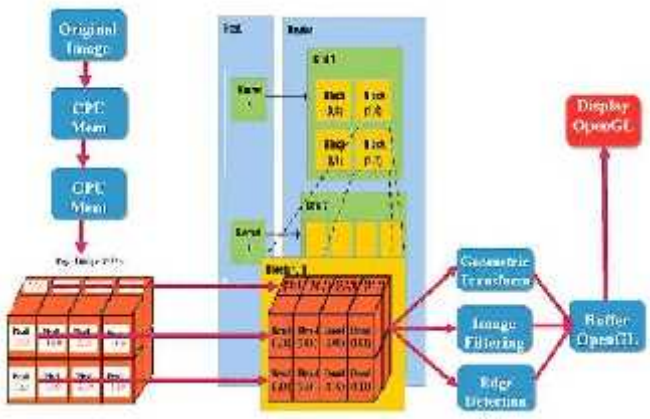


Fig. 1. Image Processing with CUDA and OpenGL

Image processing fits naturally for data parallel processing. Pixels can be mapped directly to threads and lots of data are shared between pixels [3]. In fig.1 displays the process of various image processing algorithms on GPU using CUDA and OpenGL.

USING PIXEL BUFFER OBJECTS WITH CUDA AND OPENGL FROM WINDOW CREATION TO CUDA BUFFER REGISTRATION

Before mapping the OpenGL buffer to CUDA, the following steps must be taken:

1) Create a window (OS specific).
2) Create a GL context (also OS specific).
3) Set up the GL view port and coordinate system.
4) Generate one or more GL buffers to be shared with CUDA.
5) Register these buffers with CUDA.

fig.2 illustrates these steps.

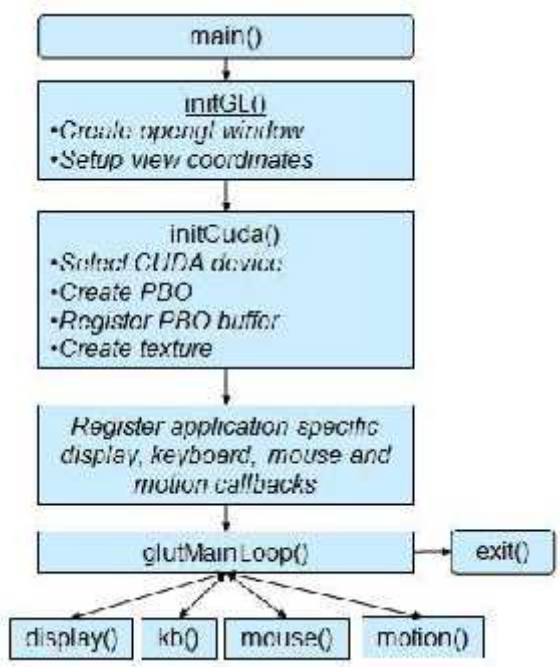


Fig. 2. CUDA Buffer Registration

USING PIXEL BUFFER OBJECTS WITH CUDA AND OPENGL FINAL STEPS TO RENDER AN IMAGE FROM A CUDA APPLICATION

To draw an image from a CUDA application requires the following steps:

1) Allocate OpenGL buffer(s) that are the size of the image.
2) Allocate OpenGL texture(s) that are the size of the image.
3) Map OpenGL buffer(s) to CUDA memory.
4) Write the image from CUDA to the mapped OpenGL buffer(s).
5) Unmap the OpenGL buffer(s).
6) Bind the texture to the OpenGL buffer [4].
7) Draw a Quad that specifies the texture coordinates at each corner.
8) Swap front and back buffers to draw to the display.

In this application, createPBO() allocates the OpenGL PBO buffer(s) with glBufferData(), thereby fulfilling step 1. Similarly, createTexture() allocates the OpenGL texture(s) specified in step 2 that can be used for rendering the image to the display. The routine display() calls the CUDA kernel that creates or modifies the data in the OpenGL buffer, then renders the new image to the screen as in fig.3.

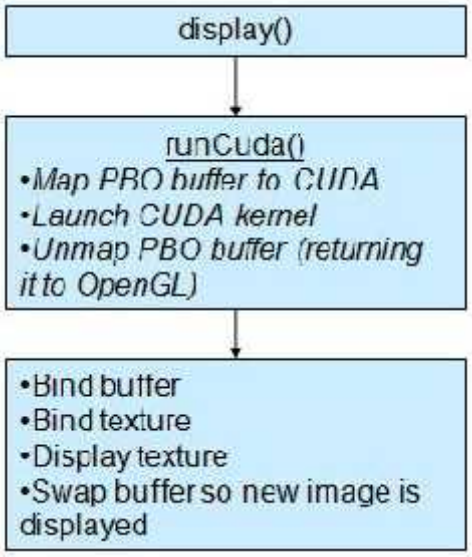


Fig. 3. Render an image from CUDA Application

IMAGE PROCESSING APPLICATION WITH CUDA

We implemented an image processing application to demonstrate the benefits of CUDA, to be more familiar with GPGPU

and create a CUDA kernel to achieve more speedup. So we implemented Sobel Edge Detection Filter.

A. Sobel Edge Detection Filter

SOBEL operator is a discrete differential operator, computing an approximation of the gradient of the image intensity function.

It is used for edge detection, filtering each pixel with a 3×3 kernel. Matrix operations calculate derivations of vertical and Horizontal changes [5].

ALGORITHM:

Sobel edge detection filter uses the two 3X3 templates to calculate the gradient value. The actual Sobel masks are shown below:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig. 4. Sobel Mask

Here in this fig.4 displays two-dimensional convolution mask."Gx" is horizontal; "Gy" is vertical operation [6].These operations then combined as for gradient magnitude as shown in Equation 1 and for gradient direction as shown in Equation 2.

$$G = \sqrt{Gx^2 + Gy^2} \quad (1)$$

$$\Theta = \arctan\frac{Gy}{Gx} \quad (2)$$

Now, consider the following 3×3 image window.

$$\begin{bmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \\ a7 & a8 & a9 \end{bmatrix}$$

where:
a1 .. a8 are the grey levels of each pixel in the filter window.

X = −1 × a1 + 1 × a3 − 2 × a4 + 2 × a6 − 1 × a7 + 1 × a9 (3)
Y = 1 × a1 + 2 × a2 + 1 × a3 − 1 × a7 − 2 × a8 − 1 × a9 (4)

Sobel gradient = $\sqrt{Gx^2 + Gy^2}$

Algo:

for R G an B components each . for(R G and B components) {

if (pixel is not one of the edge pixel) {

a1 = -1 × a1 + 1 × a3 - 2 × a4 + 2 × a6 - 1 × a7 + 1 × a9;

a2 = 1 × a1 + 2 × a2 + 1 × a3 - 1 × a7 - 2 × a8 - 1 × a9;

sourcePixel = abs(a1) +abs(a2) }

}

Sobel edge detection filter can be broken down into the following steps:

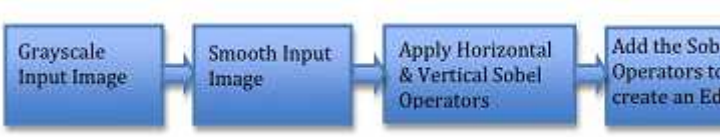


Fig. 5. Sobelfilter steps

First we need to set up the image data objects which will store the intermediate and final results, these will all be the size of the input Image [7]. There are a number of different ways to convert a color image to greyscale. But in this Program we have to first convert image in to PPM(Portable Pixel Map) and PGM (Portable Gray Map) format. Next ,do the Computation Using Sobel Operators and do the Sum of the Sobel Operators to create and Edge Image.

Procedure for create an Edge image

For Developing the Sobel Edge Detection Filter we need Graphics library (OpenGL). Various steps should be follow for this filter program.
1) First initialize OpenGL context, so we can properly set the GL for CUDA. This is necessary in order to achieve optimal performance with OpenGL/CUDA interop.
2) Use command-line specified CUDA device, otherwise use device with highest Gflops/s.
3) Kernel launch configuration.
4) Allocate Host/Device Memory.
5) Allocate and initialize an array of stream handles.
6) Create CUDA event handles.
7) Measure the time for kernel and memory copy from device.
8) Measure the time for non-streamed and streamed execution for reference. (Here asynchronously launch n streams kernels, each operating on its own portion of data)
9) Release Resources.
10) Display the various functions.
11) Free Allocated memory
12) CUDA Thread Exit

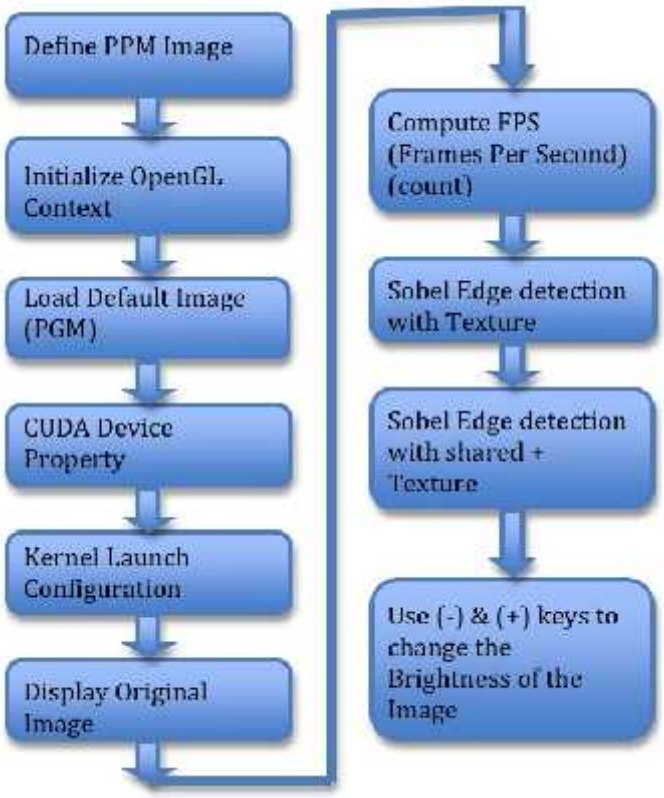


Fig. 6. Sobel Edge Detection Procedure

B. Experimental Results

Input Image: Saturn.pgm

1.Original Image (Press Key (I)) shown in fig.7.

In original Image 188 FPS(Frames Per Second) is computed.

First shared memory image shown in fig.8a, sobel edge detection computed with texture memory and number of frames are 237. Second image is shown in fig.8b, sobel edge detection computed with texture and shared memory & no of frame count is 248.

In fig.9a and fig.9b, display the more and less brightness of the image.

C. Result Analysis

Global memory is the basic problem of sobel filter. There are number of uncoalesced memory detect in this filter. So we need to optimize the filter for more speedup and less amount of uncoalesced memory.

Fig. 7. Original Filter

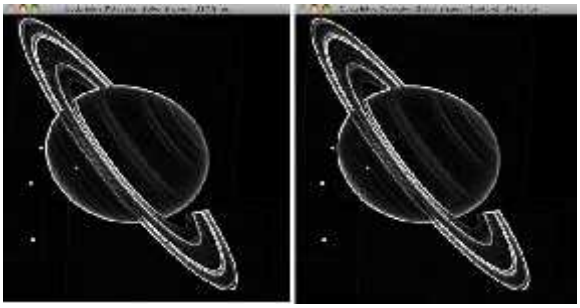


Fig. 8. Shared and Shared + Texture Image

TABLE I

Sobel Edge Detection Filter Analysis is:

Input Image	GPU Processing	CPU Processing	Speedup	Uncoalesced
128×128	45.13	301.77	6.6	2940928
256×256	215.27	578.22	2.6	16318464
512×512	302.71	574.72	1.9	0
1024×102	2942.23	2993.46	1.01	14260633

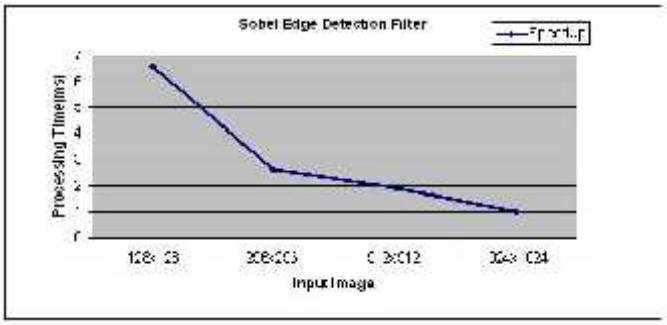


Fig. 10. Speedup of Sobel Edge Detection Filter Analysis

Original Sobel filter 512×512 input images we can get 578945024 uncoalesced memories and speedup is 1x. So we need to optimize this filter. After applying the kernel function using streams (Asynchronous calls) we can get more speedup and less uncoalesced memory used compare to original sobel filter for input Image 512×512. In this kernel function we have used 256 threads and 16 thread blocks.

In this speedup graph shown in fig.10 we can see that if input image size increases, speedup continuously decreased. Because it has some global memory limitations of GPU device. So here the results are measured for various input images and 512×512 image we can get zero (or less) uncoalesced memory.

CONCLUSION & FUTURE WORK

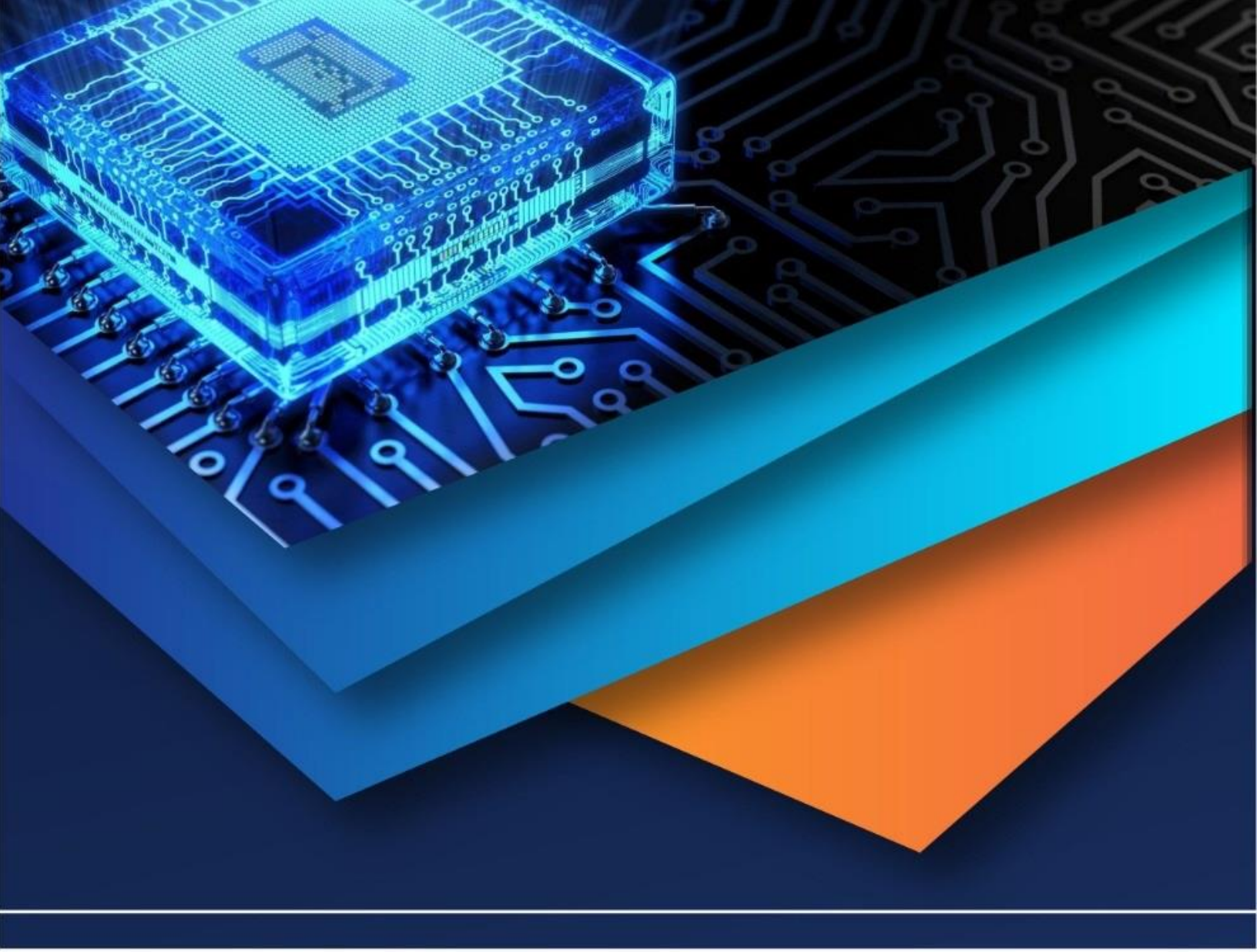
It is observed that NVIDIA graphics processing units to solve many complex computational problems in a fraction of the time required on a CPU. In sobel edge detection filter using CUDA and OpenGL displays optimal results. It counts the less number of uncoalesced memory. The capability to achieve faster speed depends upon parallelism in the program for utilizing the high no of cores for processing user friendly interfaces need to be developed so that such programmes can be developed with less efforts.

ACKNOWLEDGMENT

I would like to thank of god, my parents and friends for all their supports, which helped me reach where I am today. I am extremely thankful to all who have directly or indirectly helped me for the completion of my work. Sincere Regards to Nirma University for initiating the project.

REFERENCES

[1] Nvidia CUDA: A New Architecture for Computing on the GPU
[2] Z. Yang , Y. Zhu and Y. Pu ”Parallel Image Processing Based on CUDA”, Dept.Computer of Northwestern Polytechnical University Xian, Shaanxi, China zhuyating02@163.com
[3] F. L. M. B. S. M. Sidi Ahmed MahMoudi, Pierre Manneback,”Parallel image processing on gpu with cuda and opengl,” Computer Science Dept , faculty of Engineering , University of Mons.
[4] http://www.drdobbs.com/architecture-and-design/222600097
[5] Z.SAKA and M. KORKMAZ, ”Employing a general purpose gpu for improving the performance of cfd calculations,” tech. rep.
[6] http://www.pages.drexel.edu/
[7] http://lukewalsh.co.uk/blog/2008/06/sobel-edge-detection-in-flash. html
[8] D. C. Clarissa Tacchella,Nvidia cuda compute unified device architec- ture,”no. matr - 707827 , 708250.
[9] B. R. Neha Patil, ”Fast and parallel implementation of image processing algorithm using cuda technology on gpu hardware,” tech. rep., Depart- ment of Electrical & Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180-3590.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY