Mathematical and Computational Applications, Vol. 18, No. 3, pp. 301-312, 2013

CROPPED QUAD-TREE BASED SOLID OBJECT COLOURING WITH CUDA

Abdullah Çavuşoğlu[1], Baha Şen[1,*], Caner Özcan[2] and Salih Görgünoğlu[2]

[1]Department of Computer Engineering, Yildirim Beyazit University, Ankara, Turkey
[2]Department of Computer Engineering, Karabuk University, Karabuk, Turkey
bsen@ybu.edu.tr

Abstract-In this study, surfaces of solid objects are coloured with Cropped Quad-Tree method utilizing GPU computing optimization. There are numerous methods used in solid object colouring. When the studies carried out in different fields are taken into consideration, it is seen that quad-tree method displays a prominent position in terms of speed and performance. Cropped quad-tree is obtained as a result of the developments seen with the frequent use of this method in the field of computer sciences. Two different versions of algorithm which operate recursively on CPU and at the same time which use GPU computing optimization are used in this study. Besides, OpenGL is used for graphics drawing process. Within the setting of the study, results are obtained via CPU and GPU's, at first using Quad-Tree method and then Cropped Quad-Tree method. It is observed that GPU computing is obviously faster than CPU computing and Cropped Quad-Tree method produces rapid results compared to Quad-Tree method as a result of performance. GPU computing method boosted approximately performance by up to 20 times compared to only CPU usage; furthermore, cropped quad-tree method boosted approximately performance of algorithm by up to 25 times on average dependent on screen and object size.

Key Words-Graphics processors, Parallel processing, GPU-Computing, Cuda, Computer Graphics, Object Rendering Techniques, Image Models

1. INTRODUCTION

Developments seen in the fields where computer technology has been used require shorter durations of processing time on huge data sets for solving problems. Design of the systems operating faster becomes obligatory as a result of rapid increase in data sizes. Data processing speed of the systems becomes more important, whereas shorter response period is expected due to the data processing speed. Solutions to scientific problems, especially, to engineering problems are obtained by powerful computers running in parallel.

In the recent years, parallel computing and its applications become widespread in computer industry. Processing of data on graphics processing units (GPU) occurs as a new technology besides the use of central processing units on data processing. Although the studies carried out on GPU's are not new, GPU's are included in current computing areas as a new field. GPU's are generally optimized for computer graphics processes which require rapid calculation such as computer games and images. Despite the fact that the powers of high arithmetical calculations of GPU's mark out for a brilliant future, they include some limitations for programmers. They are almost used in every desktop pc, laptop pc, game console and mobile devices as a standard part of them.

When compared to CPU, they have higher memory bandwidth and floating point [1]. Nvidia developed CUDA (Compute Unified Device Architecture) programming model which enables software developers to use parallel computing by utilising C programming language. CUDA programming model allows programmers to use multi-threaded GPU's effectively in parallelisation. This model enables thousands of threads to run synchronously on GPU. Parallel computing is provided by the fine organization of threads, blocks and grids [2,3]. CUDA eliminates all difficulties since it creates parallelism manually. A program written with the support of CUDA is a series of a program sequence called as kernel. GPU makes this kernel parallel by duplicating it in requested numbers and running. Since CUDA is an extension of C programming language, there is no need to change their architectures in order to generally direct programs to CUDA library or make them multi-threaded [4].

There is not enough study on colouring solid objects with Cropped Quad-Tree method utilising GPU computing. But there are many studies related to this topic. An algorithm design was developed from a display of binary image series to generate a quad-tree in a study. Algorithm was carried out only one process on every pixel in image. In addition to this, when the tree-data structure is being generated, only maximum size of nodes are generated and therefore temporary nodes are not needed. Running duration of algorithm becomes equal to the numbers of pixels in the image [5]. In another study, representation of image with Quad-tree in deeper levels, in other terms, gradually decreasing sub-divisions, was studied. Within the scope of the study, an algorithm was given for superposing N quad trees in time proportional to the total number of nodes in the trees. Warnock-type algorithms were then presented for building the quad tree for the picture of the boundary of a polygon, and for colouring the interior of such a polygon [6].

In another study, relational-linear Quad-Tree approach for two dimensional spatial representation and manipulation was presented as a new approach. This approach unifies relational database models and the advantages of hierarchical data structures. Moreover, this approach offers flexible and powerful tools for spatial data structures and manipulation. Another advantage of this approach is that the rules are obviously clear and easily applicable [7]. An algorithm was presented for constructing a Quad-Tree for a binary image given its row-by-row description. Within the study, the algorithm processes the image one row at a time and merges identically coloured children as soon as possible, so that Quad-Tree which is a minimal size exists after processing the each pixel. According to the study, this method is superior to one which reads in an entire array and then attempts to build the Quad-Tree [8]. In another study, fast algorithm design operates on GPU for Quad-Tree structure was developed.

Three different implementations were realised for algorithm. These are completely GPU based implementation, CPU based sequential implementation and hybrid implementation. In hybrid implementation, first levels are constructed on CPU before data transfer to GPU in order to perform the rest of the stages. At the end of the study, it was seen that hybrid implementation provides better performance compared to others on sufficiently large datasets [9]. In another study, key factors in design and evaluation of image processing algorithms on the massive parallel graphics processing units (GPUs) using the compute unified device architecture (CUDA) programming model was studied. Within the settings of the study, a set of metrics especially

customized for image processing was proposed to quantitatively evaluate algorithm characteristics. Besides, the algorithms were carefully selected from major domains of image processing. It was seen that the speeds observed varies according to the characteristics of the algorithms. Intensive analyses were conducted to show the appropriateness of the proposed metrics in predicting the effectiveness of an application for parallel implementation [10]. A novel algorithm is presented to solve dense linear systems using CUDA. According to results of this study, GPU computation approximately worked 3 times faster than the CPU computation. This implementation provides significant performance improvement and can easily be used to solve dense linear system [11]. An implementation is proposed for quad-tree based solid object colouring using CUDA. The computation studies were evaluated for different solid objects and a better performance was obtained with GPU computing. According to results, GPU computation was 20 times faster than the CPU computation [12].

In this study, solid objects were coloured with Cropped Quad-Tree method utilising GPU computing optimization. Although, there are many methods used for colouring solid objects, when the studies carried out in different fields are taken into consideration, it is seen that Quad-Tree method displays a prominent position in terms of speed and performance. Within the setting of the study, results were obtained via CPU and GPU's, at first using Quad-Tree method and then Cropped Quad-Tree method. It was observed that GPU computing is obviously faster than CPU computing and Cropped Quad-Tree method produces rapid results compared to Quad-Tree method as a result of performance results obtained from the use of two methods.

2. MATERIALS AND METHODS

Quad-Tree and Cropped Quad-Tree methods are implemented to present solid objects. CPU and GPU computing is realized for comparisons of methods.

2.1. Presentation of Solid Objects by Using Cropped Quad-Tree Method

Quad-Tree is a tree data structure in which each internal node has four children. It is a structure which is used to organise pixels in the processes performed on images and computer graphics. Thousands, even millions of records can be stored within this structure. Each leaf node should not be obliged to contain a record but more than half of them should contain a record. Quad-Tree is a unique algorithm used in studies on locating pixels in two-dimensional image. Images are divided into quad parts and each quad is again divided into quads. They are generally classified according to data type they represent such as area, point, line and curves. In our study, we used area Quad-Tree structure which is appropriate for data type that should be represented. Partitioning of two dimensional spaces, dividing of the specific region or divided sub-regions into four equal quadrants can be represented with area Quad-Tree method. Each node in tree structure has either four children or no children [13,14,15]. Cropped Quad-Tree method is the enhanced version of Quad-Tree method. Here, the minimal screen part where the object is located is determined instead of performing operations on the entire image. Later, division operation is performed only within the window determined previously, in this way, adscititious processes are avoided. Consequently, benefit is obtained in algorithm in terms of speed dependent upon the size of the object on image. The

structure related to Cropped Quad-Tree structure that we used in the study is shown in Figure 1. On the left, the minimal screen part which can be represented with the object on image is selected and surrounded by the red dots. The representation of data structure of Cropped Quad-Tree related to the object which was selected on the screen is given on the right.

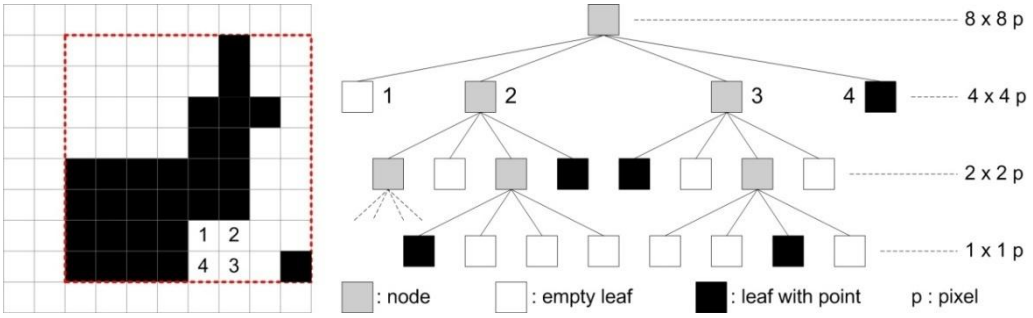


Figure 1. Data structure of cropped quad-tree

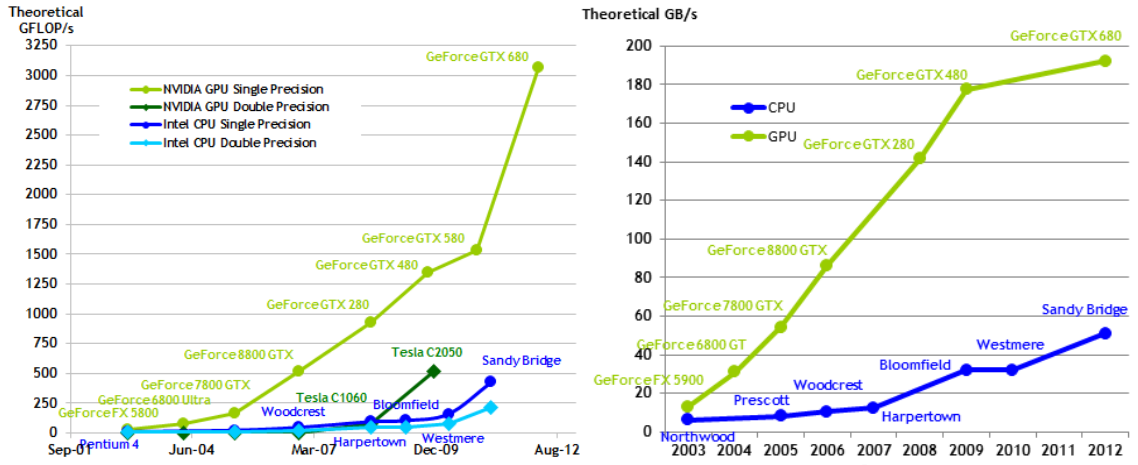
A Quad-Tree having a depth of N can be used to represent an image consisting of 2nx2n pixels where each pixel value is equal to either 0 or 1. The entire object located on image is represented by root node. If the pixels in any region are not completely 0 or 1, relevant region is again divided. In this structure, each leaf node should include a pixel block consisting of 0s or 1s. The division operation is carried out until each leaf contains only one pixel. If the region consists of pixels having same value, there is no need to divide the region again [16,17]. Application of Cropped method besides Quad-Tree method provides computational easiness. Furthermore, it is more advantageous to store only the region where the object is located in the image in the memory. As a result, it can be said that images can dynamically be represented with Cropped Quad-Tree method and this is more appropriate for image processes.

2.2. Gpu Computing and Cuda Programming Model

GPU's have been used for general programming purposes in the recent years and high speed performances have been obtained in many applications. The issue of GPU programming has not been limited with only graphics and game applications; moreover, it begins to attract the attentions of users from many different fields and also it provides many opportunities for new applications besides providing high speed in computations. GPU computational model is the use of a GPU in scientific and engineering problems. GPU computation is, in other terms, to use CPU and GPU together with in heterogeneous calculation model. Heterogeneous programming is based on the idea of independently utilising CPU and GPU which are the primary main processors of a PC according to the type of the application in order to obtain maximum efficiency from the applications. Ordered part of the application is operated on CPU and computationally predominantly computational part is operated on GPU. CPU gives the best results in serial operations as a result of many parses and random memory access. On the other hand, GPU is an expert in parallel processing with floating point operations. Briefly, the best results in serial processes are obtained via CPU and the best results in parallel processes are obtained via GPU. Heterogeneous programming is

related with the issue of utilising the appropriate processor for the appropriate process [18,19,20].

GPU floating point performance has achieved TeraFlop levels in the recent years with technological developments. Nvidia provides faster structure on GPU compared to CPU besides floating point operations per seconds and performance increases on chip bandwidths. GPU provides perfect computing power with its high parallelism and multi-threaded structure and multi-core processor architecture. Products with higher memory bandwidth have been developed by taking the demands of the users into consideration. Maximum Flops values of CPU and GPU's are given on left and memory bandwidths of CPU and GPU's are shown on the right in Figure2. Floating-point operations amount per second accessed by GPU reach to higher values rapidly. For instance, in single sensitivity computations, CPU processors can operate at maximum 475 GFLOPS level while Nvidia GeForce GTX 680 GPU processor operates at 3100 GFLOPS level. Similarly, in double sensitivity computations, CPU processors can operate at maximum 240 GFLOPS level while Nvidia Tesla C2050 GPU processor operates at 515 GFLOPS level. In the figure given, GPU bandwidth also displays higher increase compared to bandwidth of CPU. As it can be seen from these values, GPU's are rather speedy processors. Therefore, GPU's, which we selected for parallelisation, make many parts of our algorithms composed of synchronously and benefit from the advantages of high computation power of GPU. In this study, significant speed acquisitions are obtained as a result of colouring designed objects with Cropped Quad-Tree method.



(a) (b)

Figure 2. CPU and GPU comparisons (a) The maximum number of FLOPS with CPU&GPU, (b) Memory bandwidth for the CPU&GPU from 2003 to 2012 [20].

NVIDIA makes necessary changes on GPU in order to be completely programmable in scientific applications and added support for high level programming languages such as C and C++ in order to allow users use the performance obtained in various widespread platforms. This effort resulted in the development of CUDA architecture for GPU. CUDA allows users to programme GPU with various high level programming languages as software and hardware architecture. This parallel programming model allows programmers to solve problems by dividing it into sub-

problems that can be solved independently in parallelization [18]. NVIDIA gives support to users to programme GPU with C, C++, Fortran, OpenCL and DirectCompute. We developed a new algorithm design using C++ programming language in our study.

It is known that CUDA is used for calculation, data generation, and image manipulation, on the other hand, OpenGL is used to draw pixels or vertices on the screen. CUDA and OpenGL share data through common memory in the frame buffer. OpenGL buffer, texture, and render buffer objects are the OpenGL resources that may be mapped into the address space of CUDA. Sharing memory between CUDA and OpenGL can be realized by the interoperability API, as a result the particle system can be updated using CUDA, and can be rendered from the same memory using OpenGL [20]. In our study the results are displayed using OpenGL graphic functions.

2.3. CPU and GPU Design of Cropped Quad-Tree Based Solid Object Colouring

At first, the version of Cropped Quad-Tree algorithm operating on CPU was developed in the study. Later, kernel function which will operate on GPU was designed after the determination of parts which will be parallelised on algorithm. It is important to determine the intersection points of object when the screen is divided and the coordinates of the points. Furthermore, functions which will decide whether any specified point is located within a known area or not should be defined. As a result of the reasons mentioned above, the straight line denoted by xs, ys, xf and yf points also y coordinate of the point intersected by the line of screen dividing line whose x coordinate is known depicted in Figure3 are calculated by the formulae given in equation (1) and (2).

$$\frac{yf - y}{yf - ys} = \frac{xf - x}{xf - xs} \tag{1}$$

$$y = yf - (xf - x) \times (yf - ys)/(xf - xs) \tag{2}$$

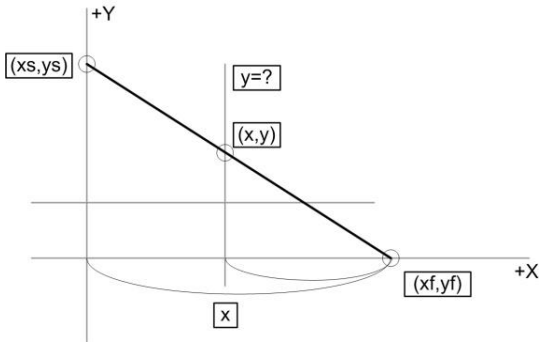


Figure 3. Graphic of function y.

Calculation of point y is performed as given in equation (1) for sample points given in Figure3 by taking triangle resemblance into consideration. If the value y is left alone in the equation, equation (2) is obtained. As a result of the use of this equation, the value of coordinate y is found out. Line can be inclined toward either right or left.

The straight line denoted by xs, ys, xf and yf points also x coordinate of the point intersected by the line of screen dividing line whose y coordinate is known depicted in Figure4 are calculated. Calculation of point x is performed as given in equation (3) by taking triangle resemblance into consideration. If the value x is left alone in the equation, equation (4) is obtained. As a result of the use of this equation, the value of coordinate x is found out.

$$\frac{xs - x}{xf - xs} = \frac{y - ys}{yf - ys} \tag{3}$$

$$x = xs - (xf - xs) \times (y - ys)/(yf - ys) \tag{4}$$

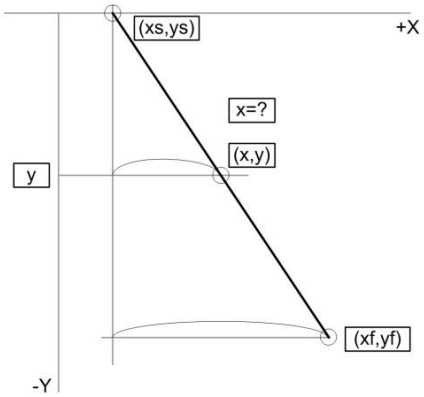


Figure 4. Graphic of function x.

A function should be written which checks whether point (xk, yk) is located within the object or not by means of point (xk, yk) obtained as a result of the division of the object by screen dividing lines in the sample rectangular object shown in Figure5 and points (sx, fx) and (sy, fy) obtained as a result of the intersection of the object by the lines forming this point. According to this, if points (xk, yk) are within the object, the function returns a true value, otherwise, the function returns a false value. If xk is among sx and sy and yk is among sy and fy, the point is located within the triangle and the function returns a true value.

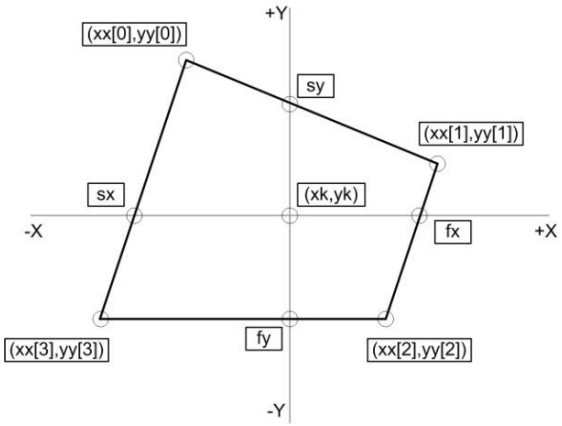


Figure 5. Sample rectangular object.

The order of the points should be in the order given in Fig. 6 when performing operations on the objects. Points which are in this order in normal circumstances change their locations after 3D rotation. The order of these points should be rearranged. An ordering function was designed for this order:

- At first, points are ordered according to y coordinate in a descending order.
- X coordinate of 0. should be lower than 1. Otherwise, points are shifted.
- X coordinate of 3. should be lower than 2. Otherwise, points are shifted.

Cropped Quad-Tree algorithm which was prepared provides benefits in terms of speed by computing the minimal screen part where the object is located and by performing the division operation only within the window determined instead of colouring the object by dividing the entire screen; furthermore, it prevents performing extra operations. The coordinates of the minimal rectangle which includes the shape are found out in order to crop the shape. The steps of the algorithm of the function which realises these operations are given below.

function CropImage{Takes Points as a Parameter}
Initialize Parameters
Set Minimum-Maximum X-Y Coordinates
Search for Min-Max X Coordinates for All Points
Search for Min-Max Y Coordinates for All Points
end function

After the crop process, screen dividing operations are initialised on the screen cropped with Quad-Tree algorithm and the object is coloured. A sample screenshot is given in Figure6.

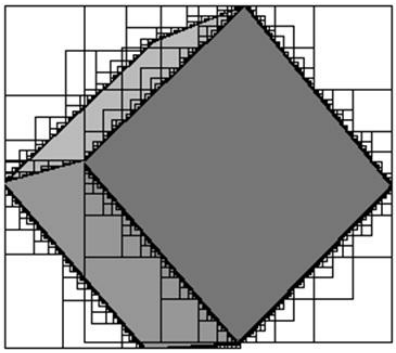


Figure 6. Sample output screen of cropped quad-tree algorithm.

Necessary parallelisation operations are carried out on the functions used and they are prepared to be able to operate on GPU after the design of the algorithm which can operate on CPU. Kernel function which will operate on GPU is designed to cover the program blocks which will operate parallel. Whether there is a vertex line within each quadrant within algorithm is checked, the minimum and maximum values for each quadrant and also the minimum and maximum values of vertex coordinates of the object

is compared with each other. Algorithm steps of the Kernel function which perform these operations are given below.

functionKernel{Takes Points and CropQuads Arrays as a Parameter}
Initialize Parameters
Initialize Crop Object
Calculate sx and fx Points
Calculate sy and fy Points
Control Given Points
end function

After the design of the Kernel function which will operate on GPU, CropQuadtree function which will operate in main function was prepared. Program blocks operating on CPU and which cannot be parallelised were included within this function. CUDA library which were included and necessary CUDA parameters were defined within main function.

function CropQuadtree{Takes Points and CroppQuads Arrays as a Parameter}
Initialize Parameters
Create Cuda Event{start, stop}
Check Cuda Device Properties{myDevice}
Sort Points of Object{cropQuads}
Memory Allocation for GPU{dev_points, dev_cropQuads}
Start Timer{start}
Memory Copy From Cpu To Gpu{dev_points, points}
Memory Copy From Cpu To Gpu{dev_cropQuads, cropQuads}
Call Kernel Function{dev_points, dev_cropQuads}
Memory Copy From Gpu To Cpu{points, dev_points}
Memory Copy From Gpu To Cpu{CropQuads, dev_CropQuads}
Crate and Display Final Image
Stop Timer{stop}
Memory Deallocation for GPU{dev_points, dev_CropQuads}
end function

Before invoking Kernel function, the transfer of data which will be used within this function was realised from CPU to GPU. After running Kernel function and performing calculation operations, the transfer of data from CPU to GPU was realised. After memory area allocated on GPU, data related to object were displayed in screen via OpenGL libraries.

3. RESULT AND CONCLUSIONS

At first the working times of CPU and then the working times of GPU were calculated utilising surfaces in different numbers prepared for Cropped Quad-Tree method. Working performances were obtained for surface numbers vary between 10 and 100 by utilising two different GPU display adapters (Gtx560ti and Quadro2000) on the same CPU. At the end of the comparison of working times, it was seen that Cropped Quad-Tree method displays a better performance compared to Quad-Tree method. This performance increase changes dependent upon the area covered by the objects on image. Our study reveals that solid objects can be coloured with Cropped Quad-Tree method in a faster manner. The results obtained at the end of the study and comparisons between performances are given in Table 1.

Table 1. CPU and GPUs working times of quad-tree and cropped quad-tree based solid object colouring method

# of Surface	**Quad-tree CpuAv. (msec)**	**Cropped quad-tree CpuAv.(msec)**	**Quad-tree Gpu1Av. (msec)**	**Cropped quad-tree Gpu1Av. (msec)**	**Quad-tree Gpu2Av. (msec)**	**Cropped quad-tree Gpu2Av. (msec)**
10	343	177	20	17	20	17
20	640	307	23	20	24	21
30	962	457	25	22	26	23
40	1253	582	30	26	32	28
50	1560	743	37	32	40	34
60	1861	884	39	34	42	36
70	2153	1019	40	34	43	37
80	2454	1154	41	35	46	37
90	2751	1273	41	35	46	38
100	3047	1404	42	35	48	39

(Gpu1: Nvidia Gtx560ti, Gpu2: Nvidia Quadro2000)

As shown in the table, the first column shows the number of different surfaces used in the study. Second and third columns show the average measurement results obtained as a result of the quad-tree and cropped quad-tree algorithm running on the CPU. The average measurement results of the quad-tree and cropped quad-tree obtained from Gtx560ti are shown in the fourth and fifth column respectively. The average measurement results of the quad-tree and cropped quad-tree obtained from Quadro2000 are shown in the sixth and seventh column respectively.

As shown in the table, the first column shows the number of different surfaces used in the study. Second column shows CPU performances of quad-tree and cropped quad-tree. Third and fourth columns show GPUs performances of quad-tree and cropped quad-tree, respectively. Cropped quad-tree performance obtained from both GPUs are shown in the fifth and sixth and seventh columns.These results also showed that GPU computing has a significant performance on the colouring cropped quad-tree based solid objects.

Table 2. CPU and GPUs performance of quad-tree and cropped quad-tree based solid object colouring method

# of Surface	Quad-tree vs. Cropped quad-tree Cpu Performance	Quad-tree vs. Cropped quad-tree Gpu1 Performance	Quad-tree vs. Cropped quad-tree Gpu2 Performance	Cropeed quad-tree Cpu vs. Gpu1 Performance	Cropped quad-tree Cpu vs. Gpu2 Performance
10	1,9x	1,2x	1,2x	10x	10x
20	2,1x	1,2x	1,1x	15x	15x
30	2,1x	1,1x	1,1x	21x	20x
40	2,2x	1,2x	1,1x	22x	21x
50	2,1x	1,2x	1,2x	23x	22x
60	2,1x	1,1x	1,2x	26x	25x
70	2,1x	1,2x	1,2x	30x	28x
80	2,1x	1,2x	1,2x	33x	31x
90	2,2x	1,2x	1,2x	36x	34x
100	2,2x	1,2x	1,2x	40x	36x

(Gpu1: Nvidia Gtx560ti, Gpu2: Nvidia Quadro2000)

In addition, graphical representation of performance of CPU and GPUs with cropped quad-tree algorithm is given in Figure7.

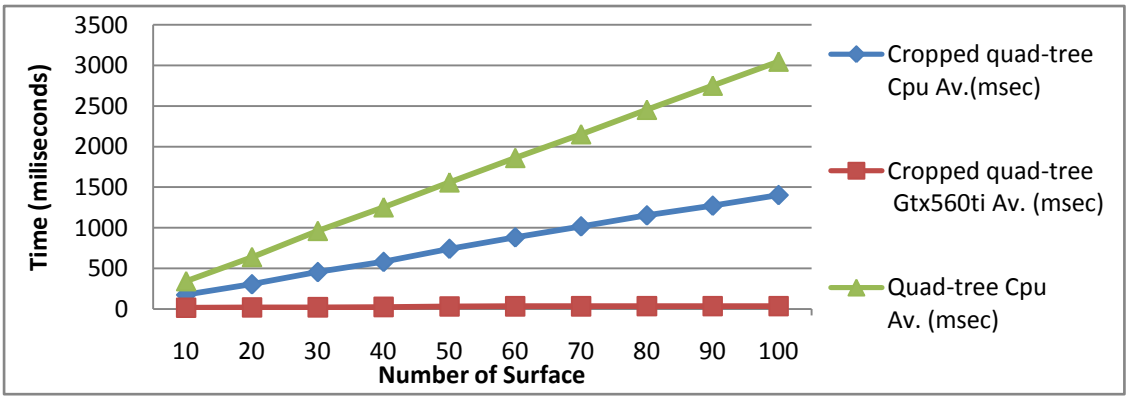


Figure 7. Performance of CPU and GPUs with cropped quad-tree algorithm

As a result, we propose an implementation for cropped quad-tree based solid object colouring using CUDA. We have tried our study on different systems that have different GPUs and CPUs. The computation studies were also evaluated for different solid objects. When we compared the results obtained from both systems, a better performance was obtained with GPU computing.

4. REFERENCES

1. J. Sanders, E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley, 2011.

2. M. Garland, S. L. Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang and V. Volkov, Parallel computing experiences with CUDA, *IEEE Micro* **28 (4),** 13-27, 2008.
3. Z. Yang, Y. Zhu and Y. Pu, Parallel Image Processing Based on CUDA, *Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, 198-201, 2008.
4. M. Akçay, B. Şen, İ. M. Orak andA. Çelik, Paralel Hesaplama ve CUDA, *6. International Advanced Technologies Symposium*, 2011.
5. H. Samet, Region representation: quad-trees from binary arrays, *Computer Graphics & Image Processing* **13 (1),** 88-93, 1980.
6. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 145-153, 1979.
7. F. Wang, Relational-Linear Quad-tree Approach for Two-Dimensional Spatial Representation and Manipulation, *IEEE Transactions on Knowledge and Data Engineering* **3 (1),** 118-122, 1991.
8. H. Samet, An algorithm for converting rasters to quadtrees, *IEEE Trans. Pattern Analysis and Machine Intelligence* **3(1),** 93-95, 1981.
9. M. Kelly andA. Breslow, Quad-tree Construction on the GPU: A Hybrid CPU-GPU Approach, Retrieved September 13, 2012 from the World Wide Web:http://www.sccs.swarthmore.edu/users/10/mkelly1/quad-trees.pdf.
10. I. K. Park, N. Singhal, M. H. Lee,S. Cho and C.W. Kim, Design and Performance Evaluation of Image Processing Algorithms on GPUs, *IEEE Transactions on Parallel and Distributed Systems* **22 (1),** 91-104, 2011.
11. C. Özcan and B. Şen, Investigation of the performance of LU decomposition method using CUDA, *World Conference on Innovation and Computer SciencesProcedia Technology* **1**, 50–54, 2011.
12. B. Şen, C. Özcan and N. A. Atasoy, An Implementation for Quad-Tree Based Solid Object Coloring Using CUDA, *AWER Procedia Information Technology and Computer Science* **1**, 2012.
13. R. A. Finkel and J. L. Bentley, Quad Trees: A data Structure for Retrieval on Composite Keys, *Acta Informatica* **4 (1),** 1-9, 1974.
14. R. Sinha, S. Samaddar, D. Bhattacharyya and T.Kim, A Tutorial on Spatial Data Handling, *International Journal of Database Theory and Application* **3(1),** 2010.
15. H. Samet, The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys* **16(2),** 187-260, 1984.
16. Quad-tree, Retrieved September 5, 2012 from the World Wide Web: http://en.wikipedia.org/wiki/Quadtree
17. G. J. Sullivan and R. L. Baker, Efficient quad-tree coding of images and video, *IEEE Trans. on Image Processing* **3**, 327-331, 1994.
18. NVIDIA CUDA C Programming Guide, Version 4.0 available as http://developer.nvidia.com/nvidia-gpu-computing-documentation.
19. J. Nickolls and W.J. Dally, The GPU computing era, *IEEE Micro* **30**, 56-69, 2010.
20. NVIDIA CUDA, Retrieved October 23, 2012 from the World Wide Web: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html