Feasibility Study of Graphics Rendering in Cloud

by

Tan Kah Meng

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Technology (Honours)
(Information and Communication Technology)

MAY 2011

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Feasibility Study of Graphics Rendering in Cloud

by

Tan Kah Meng

A project dissertation submitted to the
Information and Communication Technology Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Technology (Honours)
(Information and Communication Technology)

MAY 2011

Approved by,

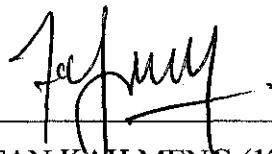
(Dr Mohamed Nordin Bin Zakaria)
Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

Approved by,

(Ms Nazleeni Samiha Binti Haron @ Baharon)
Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



TAN KAH MENG (12452)

ABSTRACT

A locally rendered graphics is restricted by the local CPU and GPU capability. However, distributing rendering process across multiple CPUs will reduce rendering time. MapReduce framework is originally developed by Google for the ease of web search applications on a large numbers of CPUs. With the application of cloud project, a MapReduce framework, graphics rendering is distributed across multiple CPU. In this research, the author studies the feasibility of using MapReduce framework on multiple CPUs to render graphics. The author even developed an interface to facilitate the display of graphics rendering in multiple frames.

ACKNOWLEDGEMENT

I feel very happy and proud to be one of the students in Universiti Teknologi Petronas and being above to develop this project as my Final Year Project. I wish to thank all personnel who have contributed directly or indirectly in accomplishment of this report, especially to my supervisor, Dr Mohamed Nordin Bin Zakaria and Ms Nazleeni Samiha Binti Haron @ Baharon, for keeping me on the right track and offering kind of encouragement when I needed it.

A high appreciation I wish to Dr Mohd Fadzil Bin Hassan, for sponsoring this project and giving his guidance, shared his knowledge and experiences in solving problems, and giving opinions throughout the project.

A full appreciation also I give to Mr. Liew Ying Wei, Low Power Embedded Processor Division manager from Intel Microelectronics for giving his concern and sponsoring items for my testing purposes. My greatest gratitude also goes to all other staffs in the department for their continuous support and the best help in assisting trainees.

I would like to express my gratitude to Andrew Goodney, Anand Raman, Kailash Gajara, and Naman Gala for providing me with their Cloud Project. This allows me to develop algorithms and my project.

Finally, my highest regards and appreciation goes back to both of my supervisors Dr Mohamed Nordin Bin Zakaria and Ms Nazleeni Samiha Binti Haron @ Baharon, who guided me in accomplishing all the other tasks such as developing the prototype, reports, presentations and etc.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background

Graphics cards play an important role in the formulation of images displayed on computer screen. Better resolution of images will require higher processing of the graphics card's capability, hence adding costs to the card itself. Amount of costs spent on graphics card alone can be exorbitant when more computer users in this world are taken into consideration.

Rather than using the graphics card itself, this project proposing images rendering via cloud computing service. At its broadest view, cloud computing is by itself location independent computing and employs scalable IT resources over the Internet; it is able to adjust and accommodate to changes on demand. Rendering the image via the cloud computing service not only saves the costs of acquiring hardware, i.e. graphics cards, it is definitely fast, convenient, easy to use from the users' end.

The aim of this project is to develop an interface to facilitate and leverage feasibility of the inherent parallelism in a 3D graphics renderer by leveraging MapReduce framework, a distributed computing framework in cloud environment. MapReduce will be further discussed in Chapter 2.

1.2 Problem Statement

Conventional graphics rendering is on local GPU and CPU. However, the process is limited by the device itself. In order to rendering a better resolutions graphics, a better graphics card is needed. If the users are able to connect to farms of computer dedicated for rendering process via cloud computing technology, it is believed the rendering process time will reduce with communication cost taken into account.

Scenario:

A user needs to do a simulation on a very complex 3D scene where huge amount processing need to be done and each transition between frames needs to be monitored. In order to process those data, user need to purchase a more powerful machine such as main frame. However, using cloud computing approach, data may be rendered in farms of computer and data is updated in order to produce the next frame.

1.3 Objectives

The objectives of this project are as follows:

- To study feasibility of rendering graphics in multiple CPUs using MapReduce framework
- To render multiple frames with existing cloud framework
- To develop an interface for viewing image

1.4 Scope of Study

This research covers creation of an interface to facilitate rendering process and building upon existing cloud framework to render multiple frames

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

Due to the emergence of 3D graphics, graphics cards are equipped with substantial processing power due to the necessity to support higher end of interactive games and applications. Graphics are being rendered locally using GPU and CPU. However by connecting local computers to farms of computer which applies MapReduce framework, graphics processes may be distributed across farms of computing to reduce rendering time. Each of the section below will discuss about cloud computing and graphic card respectively.

2.2 Cloud Computing

Cloud computing is a paradigm shift that enables scalable processing and storage over distributed, networked commodity machines. (Coombe, 2009) In this research, the resources are provided over the Internet by the public cloud, via web services. The architecture used to delivers resources to the users is Infrastructure as a Service (IaaS). Rather than purchasing graphic cards, clients instead buy those resources as a fully outsourced service.

2.2.1 Apache Hadoop

Hadoop Apache is an open source implementation of reliable, scalable, distributed computing. Hadoop is written in Java and operates on data stored in a distributed file system, usually the Hadoop Distributed File System (HDFS), which is based on the Google File System.

Hadoop is a cloud computing infrastructure that provides a virtualized interface to an arbitrarily scaled computing cluster. It automates the division of the jobs submitted by the user into sub-tasks, physical location to store data, movement of computations and

data, handling of machine failures, and all of the other details that are required for a distributed computing system to work. Hadoop instances consist of four types of processes (Hadoop): NameNode, JobTracker, DataNode, and TaskTracker.

There is one NameNode and one JobTracker in a Hadoop cluster. The NameNode manages a directory of data blocks that make up the files system namespace and access to the files stored in HDFS. The JobTracker manages jobs and coordinates sub-tasks among the TaskTrackers. A DataNode and a TaskTracker instance run on each slave machine. DataNode manage storage attached to a node and provides access to data blocks. Meanwhile TaskTracker executes tasks assigned to it by the JobTracker. Secondary NameNode provides period check-pointing and housekeeping tasks. The layout of these processes in a typical Hadoop cluster is summarized below:

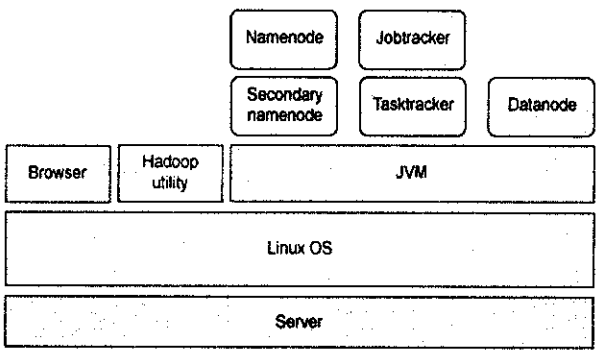


Figure 1: Pseudo-Distributed Hadoop Configuration

Hadoop is able of tolerate faults by re-executing failed tasks or tasks whose results are no longer available because of later failures, and by replicating data blocks among DataNodes. Hadoop's fault tolerance, along with its peer-to-peer bulk data transfers and largely independent tasks, allow it to scale to thousands of machines. When speculative execution is enabled, the same task may be executed on multiple nodes to increase the probability of successful and fast results.

2.2.2 MapReduce Framework

MapReduce is a programming model and implementation developed by Google that is used to process vast amounts of datasets distributed across large clusters in-parallel. (Inc.) It is highly scalable, fault-tolerant, and useful for many large-scale data processing tasks.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework takes an input key/value pair, sorts the outputs of the maps (intermediate key/value pairs) and input to the reduce tasks which eventually generates an output. Both the input and the output of the job are stored in a distributed file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

The MapReduce runtime system handles splitting the input data, scheduling map and reduce tasks, and transferring input and output data to the machines running the tasks. Jobs are managed by a master that assigns tasks to slave machines and provides the locations of intermediate values to reduce tasks. Computation on the machine where the input data is already stored is preferred in order to effectively schedule tasks and minimize network transfers. Large data transfers are performed directly between the machine where the data is stored and the machine that needs the data. Data transfers between machines on the same rack are preferred to transfers between machines that are more "distant" from each other in the network.

Map-Reduce is a framework that pools IT resources such as compute power, and storage capacity into a set of shared services that can be distributed and re-distributed as it needed across the network. This is applicable for database, system and storage administrator who seek for high performance, scalable, manageable and cost efficient

systems infrastructure. The layout of the processes in MapReduce is summarized below:

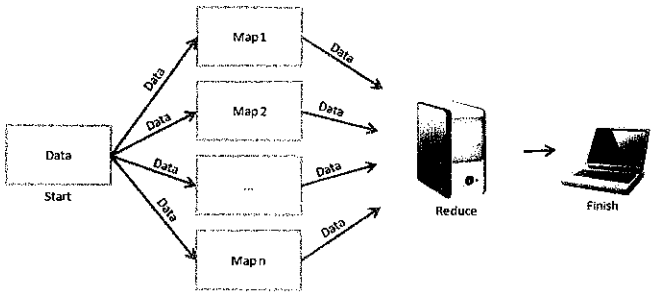


Figure 2: MapReduce Overview

2.3 Graphic Card

Graphics cards are widely used in computers nowadays. Due to the emergence of 3D graphics, graphics cards are equipped with substantial processing power due to the necessity to support higher end of interactive games and applications.

2.3.1 Graphical Processing Unit (GPU)

Aside from that, GPU is used in other domains as well. Due to its increasing ability to be programmed, it contributes to general-purpose computation. GPU is used as co-processors for CPUs, offering high performance computing, and for applications such as matrix operations, embedded system design, database, bioinformatics (Fried, June) and other less conventional ones like audio signal processing and weather forecasting. (Dinh, 2008) The special abbreviation used in relevant context will be General-Purpose Computing on GPUs (GPGPU). Not only GPUs are used in single node concept, they are also recently applied in Distributed Computing projects to further increase the processing power.

In the Map stage, a split operator will divide the input data into multiple chunks in a way that the number of chunks is equal to the number of threads. Hence, a Graphic Processing Unit (GPU) thread is

responsible with only one chunk. The runtime parameters for the Map including the number of thread groups and the number of threads per thread group are determined according to the occupancy of the GPU. This thread configuration can also be specified by the programmer himself. After the Map stage is finished, the intermediate key/value pairs are sort so that the pairs with the same key are stored consecutively.

In the Reduce stage, the split divides the sorted intermediate key/value pairs into multiple chunks which are equal to the number of threads. The thread configuration is set in a similar way as being done in the Map stage. Each chunk then is assigned to a GPU thread. Note that the key/value pairs with the same key are assigned to the same chunk. Additionally, the thread with a larger thread ID is assigned with a chunk consisting of key/value pairs of larger keys. This will ensures that the output of the Reduce stage is sorted by the key. (He, Fang, Govindaraju, Luo, & Wang, 2008) The processes can be interpreted as in Figure 3.

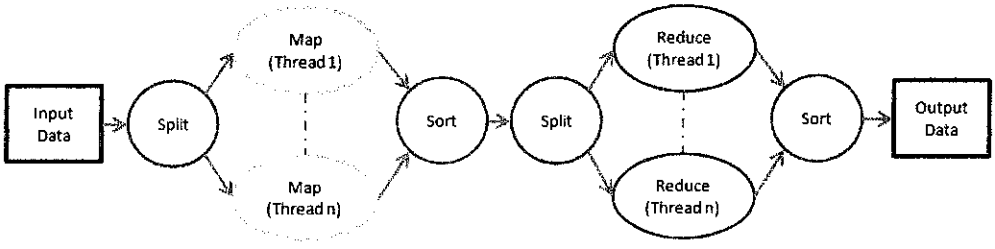


Figure 3: The Basic Flow of MapReduce on the GPU

CHAPTER 3
METHODOLOGY

3.1 Introduction

The two important frameworks which will be implemented in the cloud environment are MapReduce programming model and Hadoop framework. Both of this technology will bring great impact toward the progress of the project, such as on how we are going to build and preparing data and resources (codes and algorithms).

3.2 Previous Related Work

Based on "Mars: A MapReduce Framework on Graphics Processors" (He, Fang, Govindaraju, Luo, & Wang, 2008) it was shown that MapReduce framework, with a single GPU works up to 16 times faster than that of CPU for 6 common web applications.

The work done by Stuart et al. (2010) entitled "Multi-GPU Volume Rendering using MapReduce" has showed that MapReduce programming model has helps to attain efficiency in rendering large volume. When more GPUs are added to large clusters, the project bears the potential of performance gain for many tasks with reduced costs of GPU when they are added in bulk.

Moreover, studies done by Gajara et al. (2009) entitled "Cloud-based rendering using Hadoop" shows that MapReduce may be used as a platform for graphics algorithms. It is shown that time taken for rendering of models decreases when the number of nodes increases.

3.3 System Architecture

In general, the architecture of the overall system will be as follow.

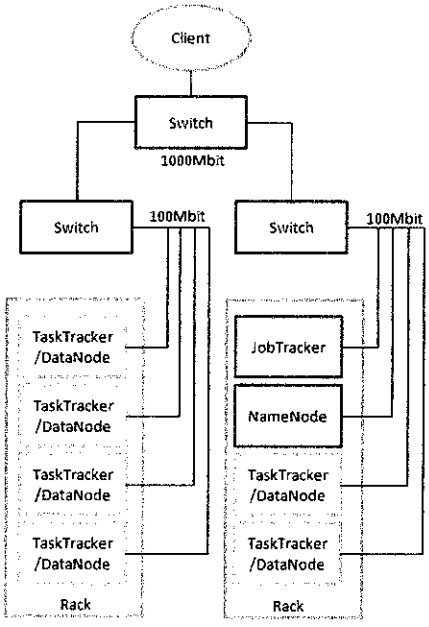


Figure 4: Hadoop Cluster

However, the architecture of the rendering process will be discussed below along with several different architectures to illustrate the difference. There are three types of architecture suitable to perform rendering in this environment. Details will be discussed below.

3.3.1. Architecture 1

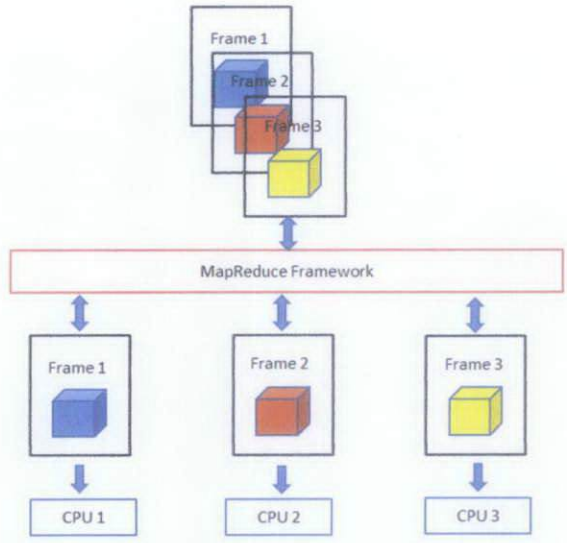


Figure 5: Distribution of Frames

In this architecture, each CPU will render one frame. MapReduce need to handle the breaking of frames through the Mapping phase and distribute it across the CPUs. The Reduce phase will be merging of frames.

The system may compile and compress a few frames before sending to users. This will reduce the network load. However, the communication cost between CPUs will be high. Rendering different frames might need different time. Therefore, it need to wait till every CPU finish rendering the frames before compiling and sending to client.

3.3.2. Architecture 2

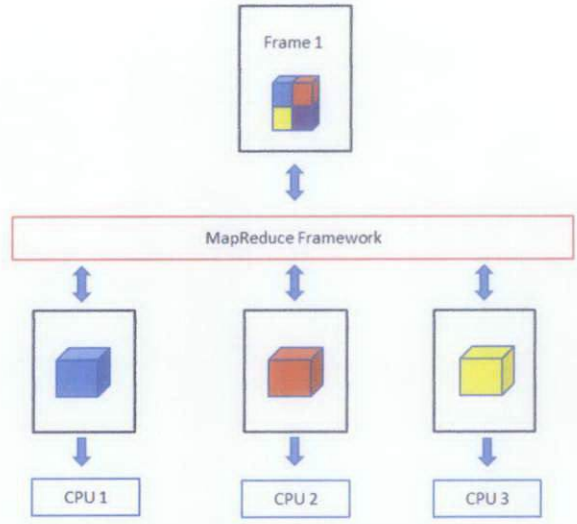


Figure 6: Distribution of Parts

In this architecture, it involves breaking of objects or portion of the frame and distributes it across CPU. MapReduce need to handle the breaking of objects through the Mapping phase and distribute it across the CPUs. The Reduce phase will be merging of portions.

The system may compile and compress a few frames before sending to users. This will reduce the communication cost but will increase the network load between the system and client.

3.3.3. Architecture 3

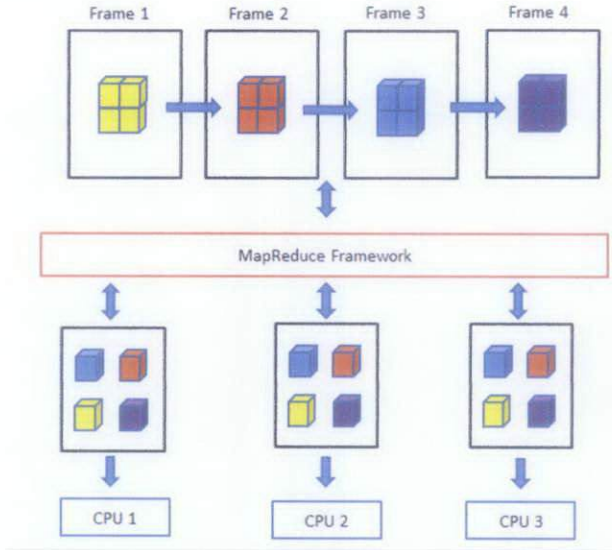


Figure 7: Distribution of Multiple Frames on Multiple Nodes

In this architecture, it involves breaking of objects or portion of the frame and distributes it across CPU. However, the process is chained where multiple frames may be processed at one time. One of the major improvements in this architecture is the system able to continuously feed the system with the data to render.

3.3.4. Summary

The architecture showed above contained its own pros and cons. 1st Architecture contains the simplicity which allows dedication of CPU to process single frame, frame does not need to be particularly distributed across the nodes. A queue system may be implemented whenever number of frames is larger than number of CPU. However, it might be very slow when there are less number(s) of CPU (E.g. 1) and need to process a huge number of frames.

In 2nd Architecture, it involves breaking of a single frame and submitted to all the processing CPU for rendering process. However, it only involves a single frame.

3rd Architecture is designed based on the 2 previous architectures. Improvements have been made such as it allows chaining of processes and rendering multiple files simultaneously through MapReduce framework. Test has been designed to demonstrate this architecture in Results and Discussion section.

3.4 System Prototyping

The development lifecycle that will be implemented for this project will be prototyping.

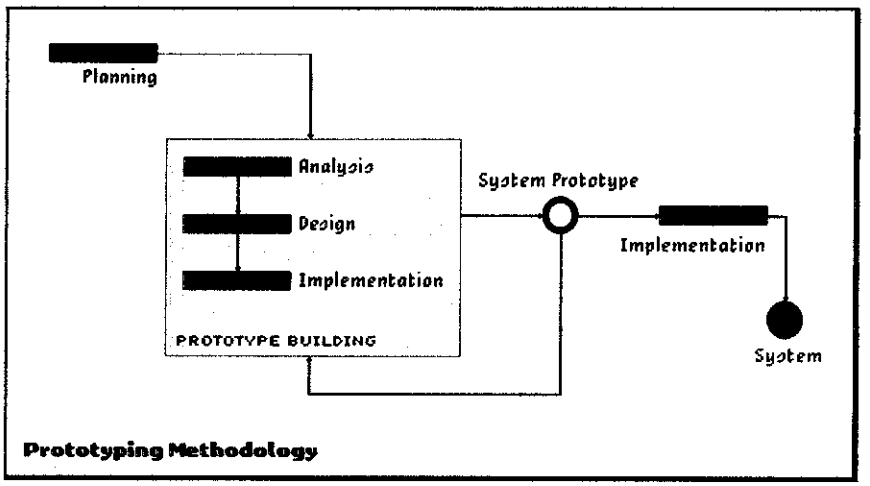


Figure 8: Prototyping

Each module is being developed and added to the main module. Below are the modules being developed:

(a) File

i. Loading of new image

ii. Loading of data (.asc)

iii. Saving of multiple frames into an archive

iv. Save a particular frame as JPEG format

v. Exiting the whole applications by killing all the process

(b) Edit

i. Undo – Remove the latest frame

(c) View

i. View in Full Screen

(d) Tool

i. Rotate image clockwise

ii. Rotate image counter clockwise

(e) Go

i. Jump to next frame

ii. Jump to previous frame

(f) Help

iii. Content

iv. About

3.5 Testing and Benchmarking

Figure 9: Black Box Testing

The prototype will be tested using Blackbox Testing approach. Each module will be tested a few rounds with different dataset along with bugs fixing to ensure quality work is produced

3.6 Tools and Equipment Required

Some of the tools and equipment which used in the project are as follows:

- Ubuntu 10.04
- Hadoop 0.20.2
- Java Development Kit 1.6.0_20
- VMware Player 3.1.2

Below is the knowledge required before the author starts this project:

- Object-Oriented Programming (Java)
- Distributed Computing
- MapReduce algorithm
- Hadoop framework
- Python

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Introduction

Figure 10: Graphics Rendering in Cloud User Interface

The project developed not only may be used as an image viewer but as well as it have the backbone of cloud computing. Data may be fed into the system to start the rendering process. Among the functionalities developed is divided into 6 groups:

(a) File
(b) Edit
(c) View
(d) Tool
(e) Go
(f) Help

Each function and each sub function will be discussed in detail below along with the screen shot of each function.

4.2 File

4.2.1. New

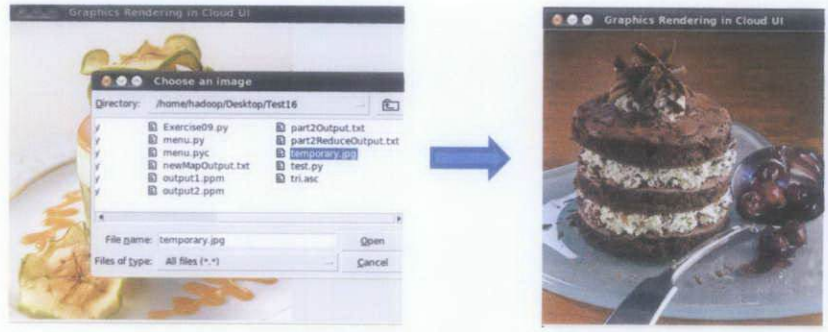


Figure 11: Load New Image

The application is able to load images (e.g. JPEG, PNG) and add it to the queue of the frames.

4.2.2. Open…

Figure 12: Load Data File

It is possible to render frames using cloud computing framework as in Architecture 3 - **Figure 7: Distribution of Multiple Frames on Multiple Nodes**.

4.2.3. Save

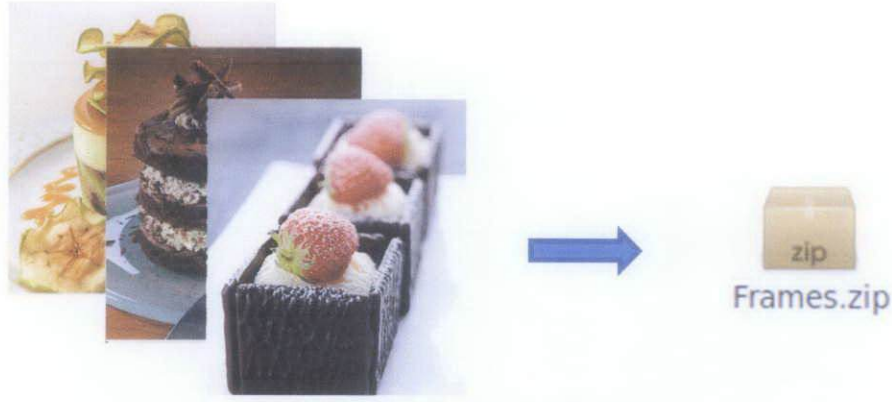


Figure 13: Archive the Frames

All the frames may be saved in an archive (.zip) to be used in the future. This function allow the user to export the whole set of data/image to other people or store it.

4.2.4. Save As

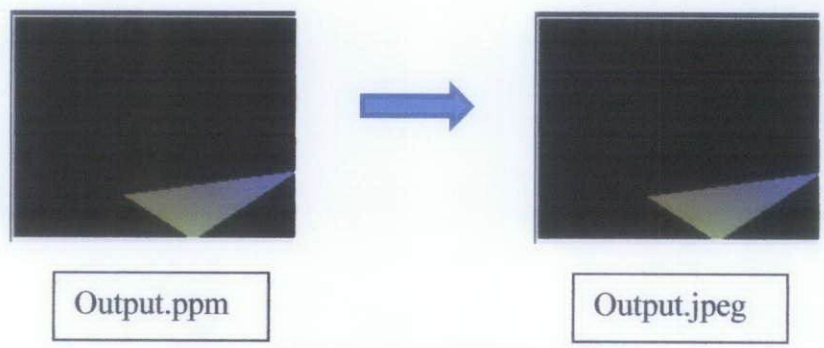


Figure 14: Save As Different Format

Image loaded using cloud framework will have .ppm format. However, this can be converted to JPEG file format. Users are able to select the directory of the file to be saved and the file name.

4.2.4. Exit

When the exit is invoked, the frames will be destroyed. The author added a shortcut for this function which is the Escape button.

4.3 Edit

4.3.1. Undo

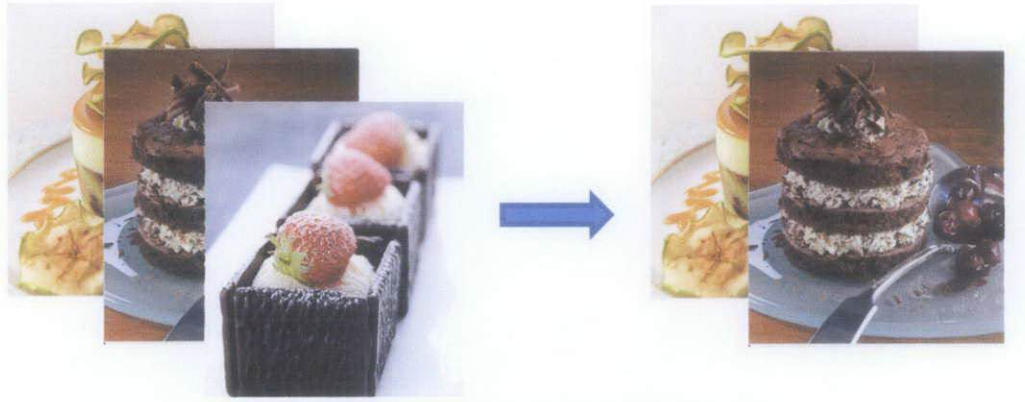


Figure 15: Remove the Last Frame

Using undo function, it allows the user to remove the latest frame added to the queue. This remove process follows the concept of LIFO (Last In First Out). However, once the image is being removed, users need to add back the frame manually.

4.4 View

4.4.1. Full Screen

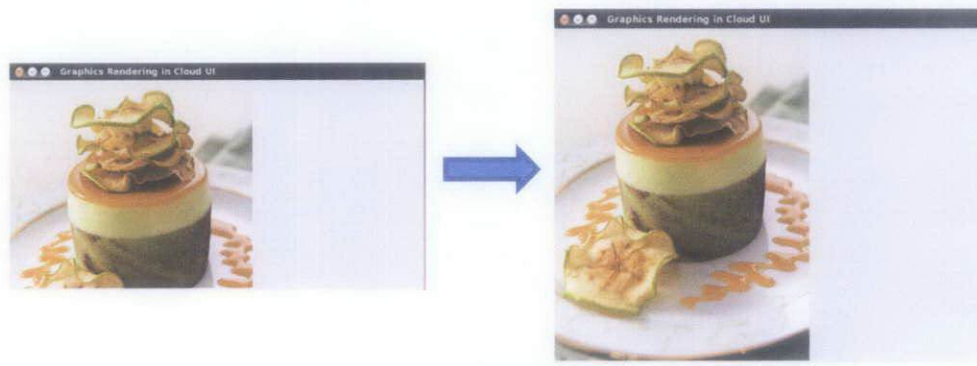


Figure 16: Viewing Full Screen

There will be time when users accidentally changed the screen size. Users may load the image in full screen mode where the size of the screen is taken to maximize the size of the panel being load.

4.5 Tool

4.5.1. Rotate Clockwise

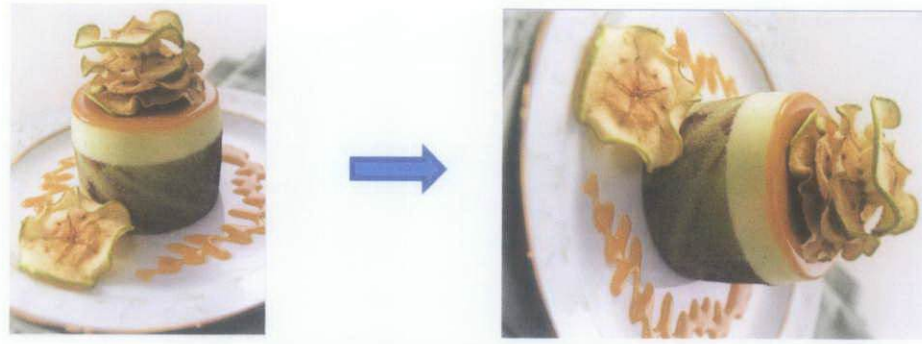


Figure 17: Rotate Clockwise

Users may rotate clockwise where it rotate 90° to the right. Image being rotated will not be saved and all the effect will be removed once the user view the next frame.

4.5.2. Rotate Counter Clockwise

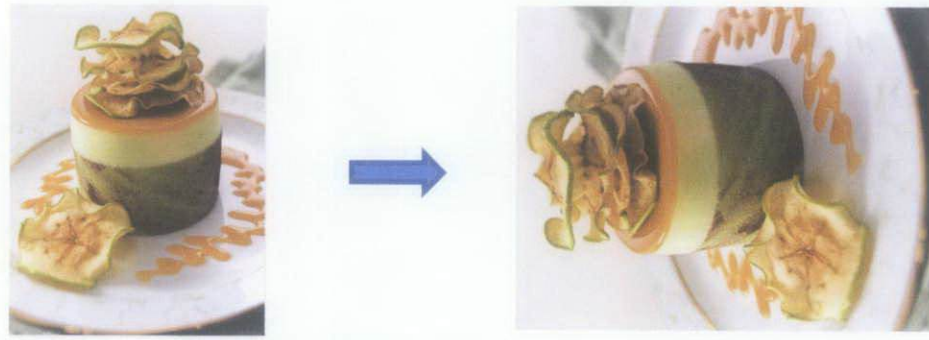


Figure 18: Rotate Counter Clockwise

The features will be the same as above except that the rotation is -90°.

4.6 Go

4.6.1. Previous Image

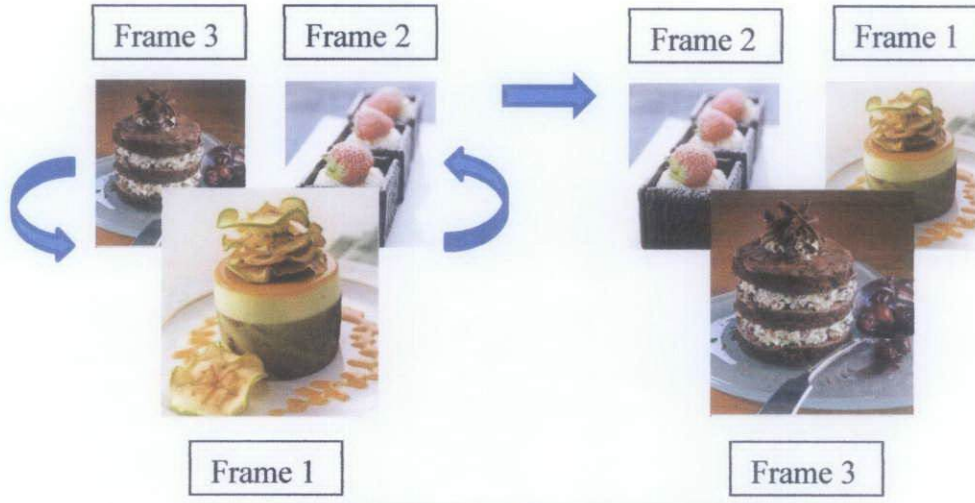


Figure 19: Goto Previous Image

Users are able to view previous images by clicking the function of previous image in Go part.

4.6.2. Next Image

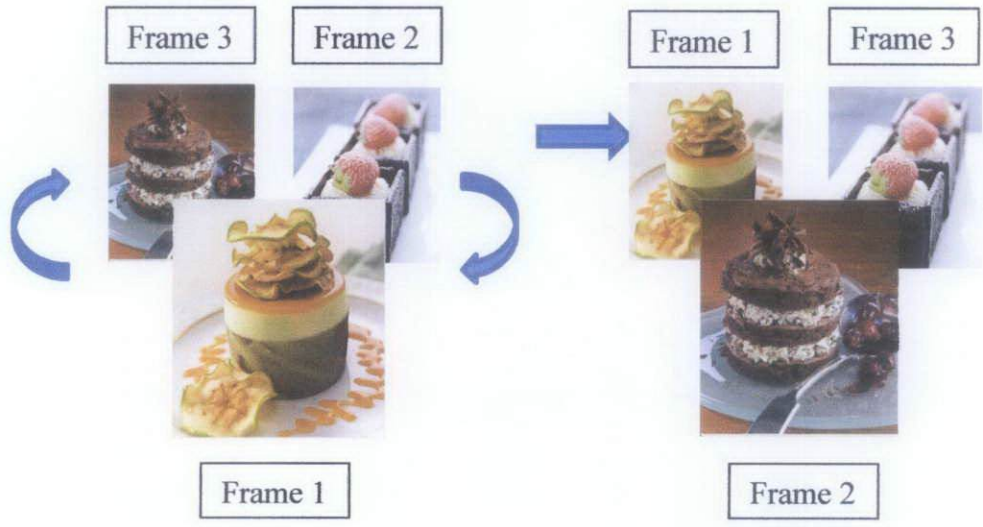


Figure 20: Goto Next Image

Users are able to view next frame by clicking the function of previous image in Go part. Moreover, a shortcut key is created as well which is the spacebar. Users may go to the next frame by tapping the spacebar.

4.7 Help

4.7.1. Content

A simple message box to describe to the users regarding on what function is available in the system.

4.7.2. About

A simple message box used to describe to the users the general information of the system such as version number and description.

4.8 Summary

This project highlights the interface of the system along with the features of distributed graphics processing across a network through MapReduce. It explores the options of performing graphics rendering with local GPU or across the cloud.

In each module developed, black box testing is applied where the functionality is tested whether it preforming as it is supposed to be. Development has been made in Python and this allows rapid prototyping of each module.

In order to simulate the architecture, we may load multiple files into the system. After 1st file is loaded, while it is still processing, 2nd file may be loaded to process. Tasks will be distributed through MapReduce Framework to all the processing nodes.

The application can be applied to students which require graphics processing application such as simulation software and etc.

CHAPTER 5
CONCLUSION

This research main purpose is to study the feasibility of rendering graphics in multiple CPUs using MapReduce framework along with developing an interface for viewing image which able to render multiple frames with existing cloud framework. These have been achieved by studying the architecture of cloud and rendering process. System has been developed to tackled the issue of conventional graphics rendering is on local GPU and CPU by using cloud computing as an approach.

The final outcome of this project is able to achieve the objectives which are as follows:

1. To study feasibility of rendering graphics in multiple CPUs using MapReduce framework
2. To render multiple frames with existing cloud framework
3. To develop an interface for viewing image

There is vast improvement can be made to the system and this research to make it better. The author currently uses existing Cloud Framework which might not be fully optimized. In the conducted study, the author does not give much attention on the performance of the system and communication cost. There is much room of improvement of this system in terms of the graphics rendering in cloud of GPUs or optimize the coding.

REFERENCES

1. Coombe, B. (2009). Cloud Computing-Overview, Advantages, and Challenges for Enterprise Deployment. *Bechtel Technology Journal , 2*(1), 4-5.
2. Dinh, M. T. (2008). GPUs - Graphics Processing Units. *Vertiefungsseminar Architektur von Prozessoren, SS.*
3. Fried, M. (2010, June). *GPGPU Architecture Comparison of ATITM and NVIDIAR GPUs.* Retrieved from Microway: http://www.microway.com/pdfs/GPGPU_Architecture_and_Performance_Comparison.pdf
4. Goodney, A., Raman, A., Gajara, K., & Gala, N. (2009). Cloud-based rendering using Hadoop.
5. *Hadoop.* (n.d.). Retrieved from http://hadoop.apache.org/
6. He, B., Fang, W., Govindaraju, N. K., Luo, Q., & Wang, T. (2008). Mars: A MapReduce Framework on Graphics Processors. *Proc. PACT.*
7. Inc., Y. (n.d.). *Hadoop Tutorial from Yahoo!* Retrieved from Yahoo!: http://developer.yahoo.com/hadoop/tutorial/module5.html
8. N. Sainath, S. M. (2010). A Framework of Cloud Computing in the Real World. *Advances in Computational Sciences and Technology, 3*(2), 175-190.
9. Shimpi, A. L., & Wilson, D. (June, 2008). *NVIDIA's 1.4 Billion Transistor GPU: GT200 Arrives as the GeForce GTX 280 & 260.* Retrieved from AnandTech: http://www.anandtech.com/show/2549/2
10. *Streamyx.* (n.d.). Retrieved from tmnet Streamyx: http://streamyx.tm/
11. Stuart, J. A., Chen, C.-K., Ma, K.-L., & Owens, J. D. (2010). Multi-GPU Volume Rendering using MapReduce. *ACM HPDC* (pp. 841-848). Chicago, Illinois: ACM.
12. Wasson, S. (January , 2010). *Intel's Core i3 and i5 dual-core processors.* Retrieved from The Tech Report: http://techreport.com/articles.x/18216

APPENDIX

Appendix 1: Configure Hadoop Distributed File System in Single Node Cluster

Appendix 2: Coding

Configure Hadoop Distributed File System in Single Node Cluster

(a) Sun Java 6

Command:

```
Download JDK1.6.0_24 from Sun Microsystem website.

sudo gedit ~/.bashrc
export JAVA_HOME=/home/hadoop/jdk1.6.0_24
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=$JAVA_HOME/lib
```

Result:

```
$ java -version
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
Java HotSpot(TM) Client VM (build 19.1-b02, mixed mode, sharing)
```

(b) Configure SSH

Install SSH:

```
$  sudo apt-get install openssh-server
```

Create empty file to store SSH Key:

```
$ mkdir ~/.ssh
$ cd ~/.ssh
$ touch authorized_keys
$ touch id_rsa.pub
$ cd ..
```

Generate SSH key:

```
$ ssh-keygen -t rsa

Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
/home/hadoop/.ssh/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
```

Append the contents of the .pub file to the correct location on the remote server:

```
hadoop@ubuntu:~$ ssh-copy-id -i /home/hadoop/.ssh/id_rsa.pub hadoop@ubuntu

The authenticity of host 'ubuntu (127.0.1.1)' can't be established.
RSA key fingerprint is 82:65:92:32:92:69:a2:ce:0c:ed:f6:9b:88:84:0c:34.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ubuntu' (RSA) to the list of known hosts.
hadoop@ubuntu's password:
Now try logging into the machine, with "ssh 'hadoop@ubuntu'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Repeat the step for localhost:

```
hadoop@ubuntu:~$ ssh-copy-id -i /home/hadoop/.ssh/id_rsa.pub hadoop@localhost
```

(c) Configure Hadoop

conf/hadoop-env.sh:

```
# The java implementation to use.  Required.
 export JAVA_HOME=/home/hadoop/jdk1.6.0_24/

# Extra Java CLASSPATH elements.  Optional.
 export HADOOP_CLASSPATH=/home/hadoop/jdk1.6.0_24/lib
```

conf/core-site.xml:

```
<property>
   <name>hadoop.tmp.dir</name>
   <value>/home/hadoop/Desktop/temporary</value>
   <description>A base for other temporary directories.</description>
</property>

<property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:54310</value>
   <description>
          Name of the default file system.  A URI whose scheme and authority determine the file system
          implementation.  The uri's scheme determines the config property (fs.SCHEME.impl) naming the
          FileSystem implementation class.  The uri's authority is used to determine the host, port, etc. for a
          filesystem.
   </description>
</property>
```

conf/mapred-site.xml:

```
<property>
   <name>mapred.job.tracker</name>
   <value>localhost:54311</value>
   <description>
          The host and port that the MapReduce job tracker runs at.  If "local", then jobs are run in-process as a
          single ma and reduce task.
   </description>
</property>
```

conf/hdfs-site.xml:

```
<property>
   <name>dfs.replication</name>
   <value>1 </value>
   <description>
          Default block replication.
          The actual number of replications can be specified when the file is created. The default is used if
          replication is not specified in create time.
   </description>
</property>
```

(d) Format the NameNode

Command:

```
$ /home/hadoop/hadoop-0.20.2/bin/hadoop namenode -format
```

(e) Test Hadoop Threads

Start:

```
$ /home/hadoop/hadoop-0.20.2/bin/start-all.sh
```

Result:

```
$ jps
11668 TaskTracker
11467 SecondaryNameNode
11524 JobTracker
11824 Jps
11169 NameNode
11319 DataNode
```

Stop:

```
$ /home/hadoop/hadoop-0.20.2/bin/stop-all.sh
```

Coding

```
import Tkinter as tk
from PIL import Image, ImageTk
import tkFileDialog

import re
import sys
sys.path.append( '/home/hadoop/Desktop/Test16/cloud/project/xform')
import xformMapper
import zipfile
root = tk.Tk()
root.title('Graphics Rendering in Cloud UI')
root.withdraw()

list_images = [ '1.jpg' ] #To load a default image
img_index = 0
rotateCounter = 0

def create_window():
        global root
        global top
        top = tk.Toplevel(root)

        w, h = root.winfo_screenwidth(), root.winfo_screenheight()

        # make the root window the size of the image
        top.geometry("%dx%d+0+0" % (w, h))

        global app
        app = tk.Frame(top)
        app.grid()

def full_screen():
        w, h = root.winfo_screenwidth(), root.winfo_screenheight()
        top.geometry("%dx%d+0+0" % (w, h))

        #global app
        app = tk.Frame(top)
        app.grid()

# pick an image file you have .bmp  .jpg  .gif.  .png
# load the file and covert it to a Tkinter image object list_images[img_index]
def loadImage():
        global image1
        image1 = ImageTk.PhotoImage(Image.open(list_images[img_index]))

        # get the image size
        w = image1.width()
        h = image1.height()
```

```
51
52              # position coordinates of root 'upper left corner'
53              x = 0
54              y = 0
55              root.geometry("%dx%d+%d+%d" % (w, h, x, y))
56
57  def renderImage():
58              global panel1
59              panel1.configure(image=image1)
60
61
62  def popupChoices(event=None):
63              global app
64
65              menu = tk.Menu(app, tearoff = 0)
66
67              menu_file = tk.Menu(menu, tearoff = 0)
68              menu_edit = tk.Menu(menu, tearoff = 0)
69              menu_view = tk.Menu(menu, tearoff = 0)
70              menu_tool = tk.Menu(menu, tearoff = 0)
71              menu_go = tk.Menu(menu, tearoff = 0)
72              menu_help = tk.Menu(menu, tearoff = 0)
73
74              menu.add_cascade(menu=menu_file, label='File')
75
76              menu_file.add_command(label='New', command=new_image)
77              menu_file.add_command(label='Open...', command=fadd_file)
78              menu_file.add_separator()
79              menu_file.add_command(label='Save', command=zip_file)
80              menu_file.add_command(label='Save As...', command=fsave_as)
81              menu_file.add_separator()
82              menu_file.add_command(label='Exit', command=fexit)
83
84
85              menu.add_cascade(menu=menu_edit, label='Edit')
86
87              menu_edit.add_command(label='Undo', command=remove_latest)
88
89
90              menu.add_cascade(menu=menu_view, label='View')
91
92              menu_view.add_command(label='Full Screen', command=full_screen)
93
94
95              menu.add_cascade(menu=menu_tool, label='Tool')
96
97              menu_tool.add_command(label='Rotate Clockwise', command=rotateClockwise)
98              menu_tool.add_command(label='Rotate Counterclockwise', command=rotateCounterClockwise)
99
100
101             menu.add_cascade(menu=menu_go, label='Go')
```

```
102
103             menu_go.add_command(label='Previous Image', command=previousImage)
104             menu_go.add_command(label='Next Image', command=nextImage)
105
106
107             menu.add_cascade(menu=menu_help, label='Help')
108
109             menu_help.add_command(label='Content', command=content)
110             menu_help.add_separator()
111             menu_help.add_command(label='About', command=about)
112
113
114             # Get the current y-coordinate of the Entry
115             ycoord = app.winfo_pointery()
116
117             # Get the current x-coordinate of the cursor
118             xcoord = app.winfo_pointerx()
119
120             # Display the Menu as a popup as it is not associated with a Button
121             menu.tk_popup(xcoord, ycoord)
122
123     def new_image():
124             global img_index, list_images
125             fileName = tkFileDialog.askopenfilename(title = 'Choose an image', filetypes=[("JPEG",".jpeg"),("All files",".*")] )
126             if fileName != None:
127                     list_images.append(fileName)
128                     nextImage()
129
130     def fadd_file():
131             #file = tkFileDialog.askopenfile(parent=root,mode='rb',title='Choose a file')
132             file = tkFileDialog.askopenfilename(title = 'Choose a file' )
133             if file != None:
134                     f = open(file, "r")
135                     text = f.read()
136                     xformMapper.main(text,len(list_images))
137                     image_name = "output"+"%i" %(len(list_images))+".ppm"
138                     print image_name
139                     list_images.append(image_name)
140
141
142     def zip_file():
143             zip_file_name = tkFileDialog.asksaveasfilename(title="Zip", filetypes=[(".zip",".zip")])
144             zout = zipfile.ZipFile(zip_file_name, "w")
145             for fname in list_images:
146                     zout.write(fname)
147             zout.close()
148
149
150     def fsave_as():
151             saveAs_file_name = tkFileDialog.asksaveasfilename(title="Save As", filetypes=[("JPEG",".jpeg"),("All
152     files",".*")])
```

```
153
154             file_name = list_images[img_index]
155             input_text = open(file_name, 'rb').read()
156             input_text = input_text[3:]
157             image_width, image_height, max_color = re.findall("(\d+)\s+(\d+)\s+(\d+)\s+", input_text)[0]
158
159             to_remove = re.findall("(\d+\s+\d+\s+\d+\s+)", input_text)[0]
160             input_text = input_text.replace(to_remove, "")
161
162             new_img = Image.new("RGB", (int(image_width), int(image_height)))
163
164             for i in xrange(int(image_height)):
165                     for j in xrange(int(image_width)):
166                     r, g, b = tuple(input_text[:3])
167                     #print ord(r), ord(g), ord(b), r, g, b
168                     input_text = input_text[3:]
169                     new_img.putpixel((j, i), (ord(r), ord(g), ord(b)))
170
171             new_img.save(saveAs_file_name)
172
173
174     def remove_latest():
175             del list_images[len(list_images)-1]
176
177     def fexit():
178             top.destroy()
179             root.destroy()
180
181     def nextImage():
182             global img_index, list_images
183             global rotateCounter
184             rotateCounter = 0
185
186             if len(list_images)-1 > img_index:
187                     img_index += 1
188             else:
189                     img_index = 0
190             loadImage()
191             renderImage()
192
193     def previousImage():
194             global img_index, list_images
195             global rotateCounter
196             rotateCounter = 0
197
198             if len(list_images)-1 > img_index:
199                     img_index -= 1
200             else:
201                     img_index = 0
202             loadImage()
203             renderImage()
```

```
204
205
206 def rotateClockwise():
207         global rotateCounter
208         rotateCounter -= 90
209
210         image = Image.open(list_images[img_index])
211         imRotate = image.rotate(rotateCounter)
212         filename = "temporary.jpg"
213         imRotate.save(filename)
214
215         list_images.insert(img_index, filename)
216
217         loadImage()
218         renderImage()
219         del list_images[img_index]
220
221 def rotateCounterClockwise():
222         global rotateCounter
223         rotateCounter += 90
224
225         image = Image.open(list_images[img_index])
226         imRotate = image.rotate(rotateCounter)
227         filename = "temporary.jpg"
228         imRotate.save(filename)
229
230         list_images.insert(img_index, filename)
231
232         loadImage()
233         renderImage()
234         del list_images[img_index]
235
236 def content():
237         root = tk.Tk()
238         root.title('Content')
239         tk.Message(root, text="The Basic Functionalities: \n"
240                 "File: New, Open, Save, Save As, Exit\n"
241                 "Edit: Undo\n"
242                 "View: Full Screen\n"
243                 "Tool: Rotate Clockwise, Rotate Counterclockwise\n"
244                 "Go: Previous Image, Next Image\n"
245                 ).pack(padx=100, pady=50)
246         root.mainloop()
247
248 def about():
249         root = tk.Tk()
250         root.title('About')
251         tk.Message(root, text="Graphics Rendering in Cloud UI \n"
252                 "Version 9.0\n"
253                 "The system is designed by Gary Tan\n"
```

```
                    "The developer believes that graphics rendering will be rendered in the Internet instead of locally in the
future"
                    ).pack(padx=100, pady=50)
        root.mainloop()


def key_escape(evt):
        fexit()

def key_space(evt):
        nextImage()


create_window()

root.bind_all('<Escape>', key_escape)
root.bind_all('<Button-3>', popupChoices)
root.bind_all('<space>', key_space)
# root has no image argument, so use a label as a panel

loadImage()
global panel1
panel1 = tk.Label(app, image=image1)
panel1.pack(side='top', fill='both', expand='yes')
panel1.image = image1

# save the panel's image from 'garbage collection'
renderImage()
# start the event loop
root.mainloop()
```