PRODUCT ARRANGEMENT FOR LOGISTICS SYSTEM

MATHUS MANINOI

RUJIRADA WATTANATANASUP

ITTIPAT CHANGSRI

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

IN COMPUTER SCIENCE (INTERNATIONAL PROGRAM)

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2011

Title	Product Arrangement for Logistics System	
Students	Mr. Mathus Maninoi	Student ID: 51051179
	Ms. Rujirada Wattanatanasup	Student ID: 51051214
	Mr. Ittipat Changsri	Student ID: 51051219
Degree	Bachelor of Science	
Program	Computer Science (International Program)	
Year	2011	
Advisor	Dr. Warangkhana Kimpan	

ABSTRACT

Nowadays, business about transportation expands rapidly and there are many types of vehicle model. However, the most frequently use of the transportation types to deliver the items is truck. In order to make it convenient in delivery products for a small business, the goal of developing this system is to satisfy the user's need and life style in daily life. The system provides fast, convenient, and cost-effective in every investment. Therefore, either large or small businesses can use many types of vehicle for delivery the products to increase value of businesses.

This special problem focuses on loading the merchandises into the vehicles. In order to bring maximum benefit and reduce the business cost. The principle of Bin-packing algorithm was applied to simulate how to fill the material into the vehicles. Also, the system was developed and simulated by computer program to support the best solution of loading material into the vehicles.

ACKNOWLEDGEMENTS

We would like to express my sincere thanks to my special problem advisor, Dr.Warangkhana Kimpan for her invaluable help and constant encouragement throughout the course of this project. We would not have achieved this far and this special problem would not have been completed without all the support that we have always received from her. We would like to thankful our friends and other person for suggestion, information and their help.

Finally, we most gratefully acknowledge to our parents for all their support and comfort throughout the period of this special problem.

Mathus Maninoi

Rujirada Wattanatanasup

Ittipat Changsri

April 2012

Contents

	Page
Abstract	I
Acknowledgements	II
Contents	III
List of Tables	VI
List of Figures	VII
Chapter 1 Introduction	**1**
1.1 Importance and Cause of Problem	1
1.2 Purpose of the Special Topic	1
1.3 Coverage of the Special Topic	1
1.4 Expected benefits	2
1.4.1 Benefits to the system developers	2
1.4.2 Benefit to the users	2
1.5 Implementation procedures	2
1.6 Project Planning	3
1.7 Equipment to be used in Special Problem	4
1.7.1 Description of the computer equipments	4
1.7.2 Details of the Program	4
Chapter 2 Background	**5**
2.1 Logistics System	5
2.2 Bin Packing Problem	6

Contents (Cont.)

	Page
2.2.1 Problem formulation	6
2.2.2 3D Best Fit Algorithm	8
2.2.3 3D First Fit Decreasing Algorithm	8
2.3 OpenGL	9
2.3.1 What Is OpenGL?	9
2.3.2 A Smidgen of OpenGL Code	12
2.3.3 OpenGL Command Syntax	15
2.3.4 Include Files	17
2.3.5 GLUT, the OpenGL Utility Toolkit	18
Chapter 3 System Design	**19**
3.1 Use Case Diagram	19
3.2 Activity diagram	20
3.2.1 Activity Diagram of Add Container and Add Product	20
3.2.2 Activity Diagram of View Arrangement Result	21
3.3 Control Hierarchy Diagram	22
3.4 Graphic User Interface	23
3.4.1 GUI in the 1st page of the program (Main Page)	23
3.4.2 Result page	24

Contents (Cont.)

	Page
Chapter 4 Experimental Results	**25**
4.1 Results and Experimental Results	25
4.1.1 Case 1	25
4.1.2 Case 2	30
4.1.3 Case 3	34
4.1.4 Case 4	38
4.1.5 Case 5	42
4.1.6 Conclusion	46
Chapter 5 Conclusion and Problem	**47**
5.1 Conclusion	47
5.2 Problems	47
5.3 Limitation	48
5.4 Future work	48
References	**49**
Appendices	**50**
Appendix A	51
Appendix B	55

List of Tables

Table	Page
2.1 Command Suffixes and Argument Data Types	15
3.1 Box information	23
4.1 User's requirement in case 1	25
4.2 User's requirement in case 2	30
4.3 User's requirement in case 3	34
4.4 User's requirement in case 4	38
4.5 User's requirement in case 5	42
4.6 Case conclusion	46

List of Figures

Figure	Page
1.1 Special problem Gantt chart	3
1.2 Special problem Gantt chart (continue)	4
2.1 The example of a distributed management for warehouses abroad	6
2.2 White Rectangles on a Black Background	13
3.1 Use Case Diagram for Product Arrangement Project	19
3.2 Activity diagram of Add Container and Add Product	20
3.3 Activity diagram of View Arrangement Result	21
3.4 Control hierarchy diagram	22
3.5 Main page	23
3.6 Result page	24
4.1 The result of arrangement in case 1	26
4.2 The graphical result of the First-Fit algorithm of case1	27
4.3 The graphical result of the First-Fit Decreasing algorithm of case1	28
4.4 The graphical result of the Last-Fit algorithm of case1	29
4.5 The graphical result of the First-Fit algorithm of case2	31
4.6 The graphical result of the First-Fit Decreasing algorithm of case2	32
4.7 The graphical result of the Last -Fit algorithm of case2	33
4.8 The graphical result of the First-Fit algorithm of case3	35
4.9 The graphical result of the First-Fit Decreasing algorithm of case3	36
4.10 The graphical result of the Last -Fit algorithm of case3	37
4.11 The graphical result of the First-Fit algorithm of case4	39

List of Figures (Cont.)

Figure	Page
4.12 The graphical result of the First-Fit Decreasing algorithm of case4	40
4.13 The graphical result of the Last -Fit algorithm of case4	41
4.14 The graphical result of the First-Fit algorithm of case5	43
4.15 The graphical result of the First-Fit Decreasing algorithm of case5	44
4.16 The graphical result of the Last -Fit algorithm of case5	45
A.1 Folder GLUT	52
A.2 Move glut.h	53
A.3 Move glut32.lib	53
A.4 Move glut32.dll	54
B.1 Microsoft Visual Studio 2008 Professional Edition product	57
B.2 Windows form to start the installation VS2008	57
B.3 Windows form to copy files into temporary folder	58
B.4 The welcome setup wizard page	59
B.5 The welcome setup wizard page in next step	60
B.6 The list of required components need to be installed	61
B.7 The features type to install program	62
B.8 Full type to installation	63
B.9 Component to be installed	64
B.10 Components to be installed (2)	64
B.11 Components to be installed (3)	65
B.12 Components to be installed (4)	65
B.13 Finished page to installed	66

List of Figures (Cont.)

Figure	Page
B.14 Dialog to restart your computer	67
B.15 The Windows Start menu for Visual Studio 2008	67
B.16 Windows form to choose default environment	68
B.17 Windows form to configuring the develop environments	69
B.18 Start page of Visual Studio 2008	69

Chapter 1

Introduction

This chapter explains the origin of the problem and the solution we envision. It also covers the scope of this special problem, the benefits of creating this system and the plan to complete the project.

1.1 Importance and Cause of Problem

Recently, the business of logistics system is widely expanded thus many delivery businesses are established. The idea of the special problem focuses on developing a program that helps companies to sort and arrange the stuffs or the products in the containers and makes the most efficiency for each trip of the delivery.

Therefore, this special problem proposes the adaptation of using Bin-packing algorithm solution knowledge in order to sort the merchandises in the containers. Moreover, the program shows the result in 3D the products by using OpenGL.

1.2 Purpose of the Special Problem

To create a program that can calculate the best way to store the merchandises into the containers based on 3D-Bin algorithm in the logistics system.

1.3 Coverage of the Special Problem

1) Create new simulated program that lets users indicate the property of the storage and the delivery order.

2) Make the program easy to use by using the 3D graphic.

1.4 Expected benefits

1.4.1 Benefits to the system developers

1) Learn and gain experiences from programming.

2) Understand about Bin-packing algorithm, logistic, and OpenGL.

1.4.2 Benefit to the users

Provide the best method for users to arrange the products by considering the delivery order and the efficiency of the providing space.

1.5 Implementation procedures

1) Study the Bin-packing algorithm along with C++ programming and OpenGL.

2) Design the functions and the features inside the application and users interface.

3) Code and implement the system.

4) Test the implemented system and eliminate a problem regarding the simulation.

5) Document the special problem report and the users' manual.

1.6 Project Planning

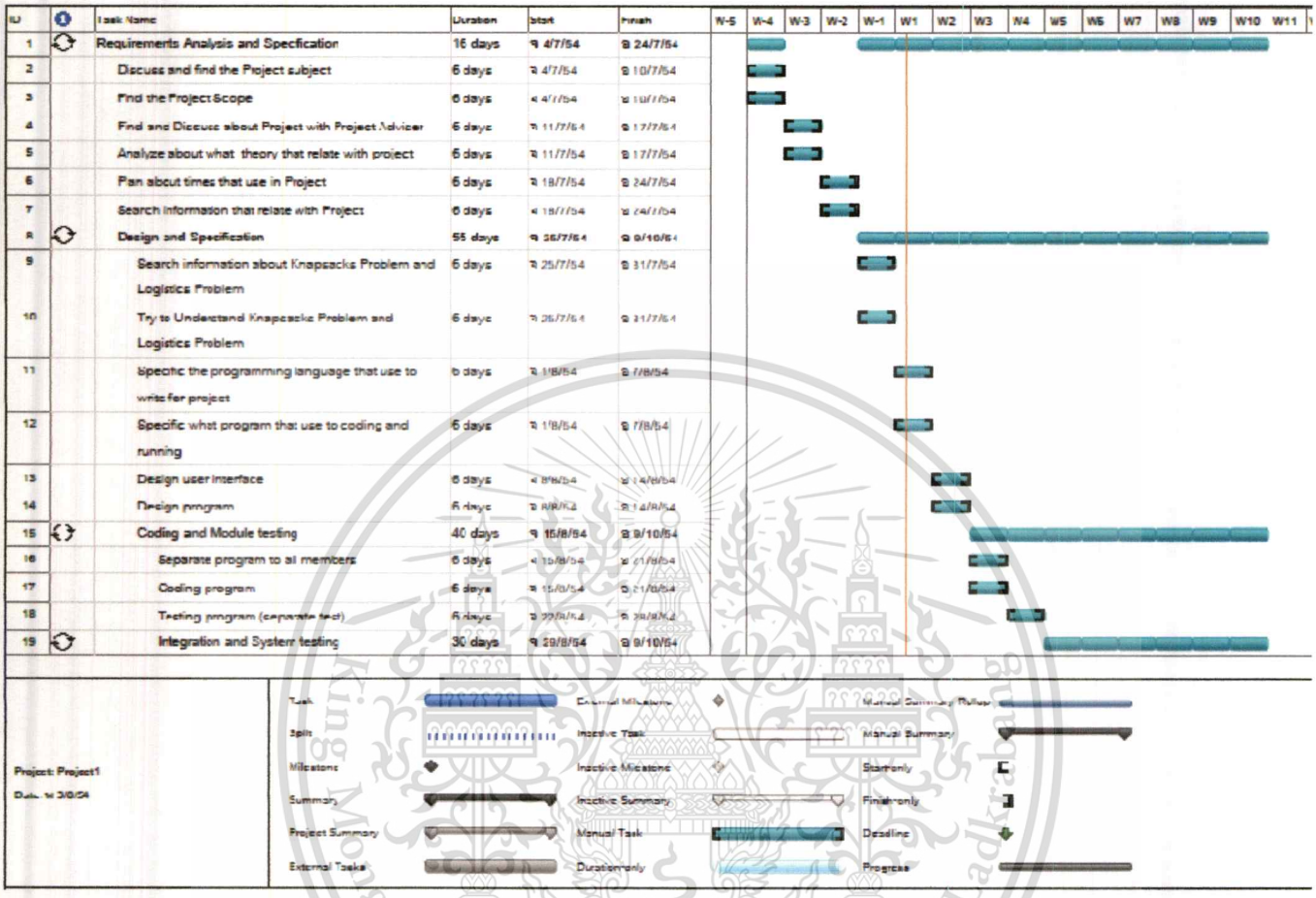


Figure 1.1 Special problem Gantt chart

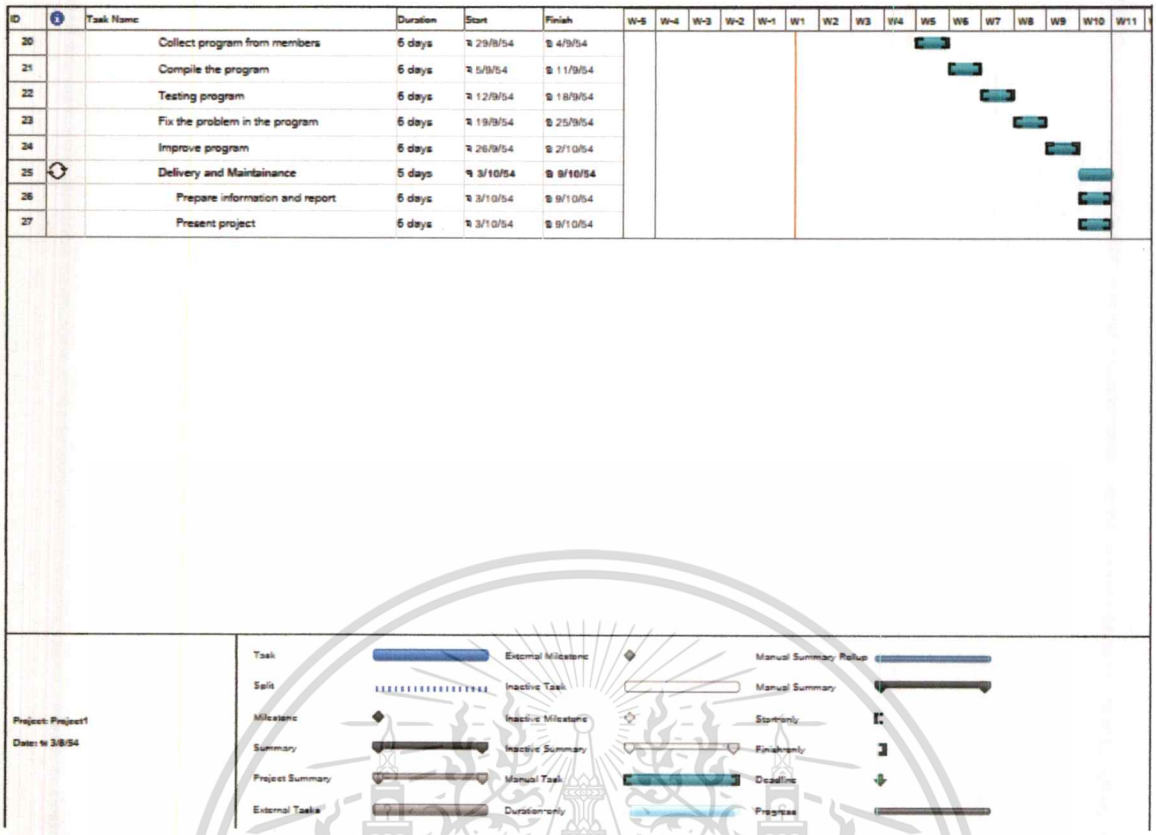


Figure 1.2 Special problem Gantt chart (continue)

1.7 Equipment to be used in Special Problem

1.7.1 Description of the Computer Equipments

Computer: minimum specification required as

1) CPU :Intel Core 2 Duo or higher
2) GPU: (graphic card) ATI Radeon HD 2400 XT
3) Ram: Up to 2GB
4) Hard disk: Up to 250 GB
5) Operation System : window 7

1.7.2 Details of the Program

1) Microsoft Visual C# or any C# compilers with Common Language Infrastructure and .NET class library up to .NET 2.0.
2) OpenGL compiler with the GLU, GLUT, GLUI, and SDT support libraries.

Chapter 2

Background

This chapter will cover the basic of Bin-packing algorithm, the relationship among the Bin-packing algorithm, Logistics system, and OpenGL. It represents how to integrate between the Bin-packing algorithm and Logistics system.

2.1 Logistics System

Logistics is a planning system in the industrial mechanization that supposed to model, analyze and optimize the whole supply chain. There are many particular problems exist in a significant progress in robot development and individual assumptions.

Logistics is a very important part of a storehouse management system. There are two advantages of logistics system; the problems can be shown in number and it can compare the different approaches to some benchmarks. Moreover, the industrial problems can be devised as bin-packing problems such as cargo loading, project selection, budget control, and etc.

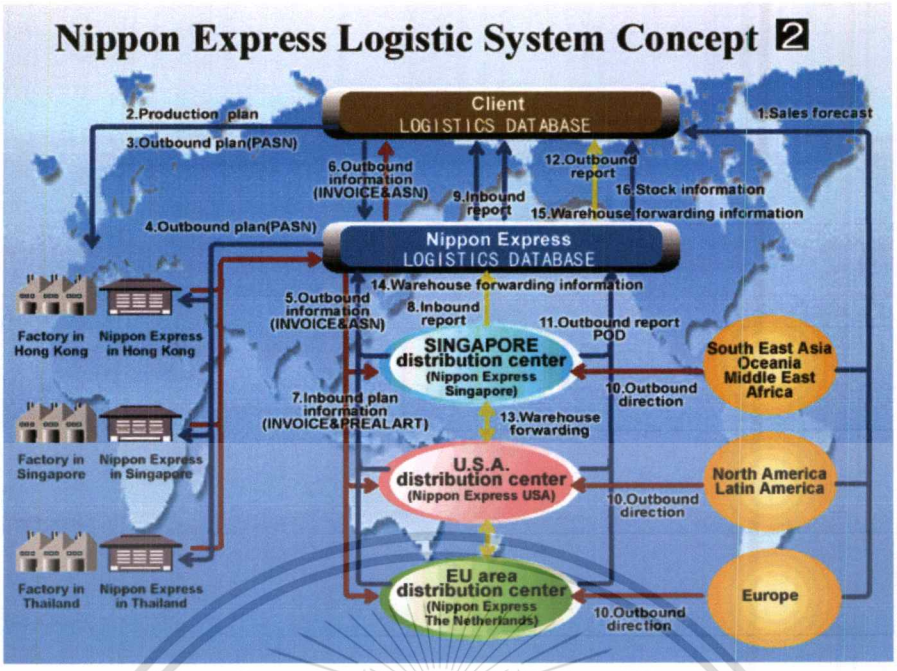


Figure 2.1 The example of a distributed management for warehouses abroad

2.2 Bin Packing Problem [1]

2.2.1 Problem Formulation

In order to find the solution for a bin b we assume without losing generality that

$$\sum_{i \in b} w_i \leq W \tag{1}$$

$$\sum_{i \in b} h_i \leq H \tag{2}$$

and

$$\sum_{i \in b} d_i \leq D \tag{3}$$

As such it is correct to conclude that

$$\sum_{i \in b} w_i \bullet h_i \bullet d_i \leq W \bullet H \bullet D \tag{4}$$

So for each bin b we intend to minimize the wasted volume given by

$$W \bullet H \bullet D - \sum_{i \in b} w_i \bullet h_i \bullet d_i \tag{5}$$

Bin packing is an NP-Hard problem, suggests that an exhaustive search for the optimal solution is in general computationally intractable, and also there is thus unknown real computationally feasible optimal solution method for the problem. Most popular method is heuristic solution methods: Items are packed one at a time with no backtracking (once an item is packed it is not repacked). The choice of an item to be packed can be done by using formal logic derived from one of the following packing algorithms.

1) First-Fit [2]

A resource allocation scheme (usually for memory). First-Fit fits data into memory by scanning from the beginning of available memory to the end, until the first free space which is at least big enough to accept the data is found. This space is then allocated to the data. Any leftover becomes a smaller, separate free space.

2.) First-Fit Decreasing

Almost the same as First-Fit except that the items are first sorted in decreasing order before being packed.

3.) Last-Fit

Packs unassigned item into last bin with enough space. Searching is similar to First-Fit but in the reverse order of bins. If there is no such bin, assign item into new bin.

4.) Best-Fit

The Best-Fit algorithm packs an item in a bin, which is the fullest among those bins in which the item fits. More specifically; items are packed one at a time in given order. To determine the bin for an item, first determine set ***B*** of containers into which the item fits. If ***B*** is empty, then start a new bin and put the item into this new bin. Otherwise, pack the item into the bin of ***B*** that has least available capacity.

2.2.2 3D Best-Fit Algorithm

Decide on a packing direction. Each bin has three directions in which to pack, a width (or x) direction, a height (or y) direction, a depth (or z) direction. Only one bin will be place at a time. We first choose a ***pivot*** point. The pivot is an (x, y, z) coordinate which represents a point in a particular 3D bin at which an attempt to pack an item will be made. The back lower left corner of the item will be placed at the pivot. If the item cannot be packed at the pivot position then it is rotated until it can be packed at the pivot point. If item still cannot be packed at the pivot point, then we move on to pack other items and add the unpacked item to a list of items that will be packed after an attempt to pack the remaining items is made. The first pivot in an empty bin is always (0, 0, 0).

2.2.3 3D First-Fit Decreasing Algorithm

To pack an item one has to first decide on a packing direction. The longest side of the bin corresponds to the packing direction. Then rotate each item such that the longest side of this item is the side which is the packing direction, i.e. if we are packing by width then we want the longest side of the item to be the item's width. For example, if the packing direction is the width and the current height of the item is longer than its width, then rotate the item. After performing the rotation(s), if the item cannot fit into the bin (i.e. one or more of the dimensions of the items exceed the bin's corresponding dimension) then we rotate the item until the *second* longest side of this item is the side which corresponds to the packing direction. After performing the

rotation(s), the item cannot fit into the bin then we rotate the item until the *third* longest side of this item is the side which corresponds to the packing direction. Next sort the items in decreasing order of the width, the height or the depth depending on packing direction.

2.3 OpenGL

2.3.1 What Is OpenGL? [3]

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining users input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you are using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

Now that you know what OpenGL *doesn't* do, here's what it *does* do. Take a look at the color plates - they illustrate typical uses of OpenGL. They show the scene on the cover of this book, *rendered* (which is to say, drawn) by a computer using OpenGL in successively more complicated ways. The following list describes in general terms how these pictures were made.

- "Plate 1" shows the entire scene displayed as a wireframe model - that is, as if all the objects in the scene were made of wire. Each line of wire corresponds to an edge of a

primitive (typically a polygon). For example, the surface of the table is constructed from triangular polygons that are positioned like slices of pie.

Note that you can see portions of objects that would be obscured if the objects were solid rather than wireframe. For example, you can see the entire model of the hills outside the window even though most of this model is normally hidden by the wall of the room. The globe appears to be nearly solid because it's composed of hundreds of colored blocks, and you see the wireframe lines for all the edges of all the blocks, even those forming the back side of the globe. The way the globe is constructed gives you an idea of how complex objects can be created by assembling lower-level objects.

- "Plate 2" shows a *depth-cued* version of the same wireframe scene. Note that the lines farther from the eye are dimmer, just as they would be in real life, thereby giving a visual cue of depth. OpenGL uses atmospheric effects (collectively referred to as fog) to achieve depth cueing.
- "Plate 3" shows an *antialiased* version of the wireframe scene. Antialiasing is a technique for reducing the jagged edges (also known as *jaggies*) created when approximating smooth edges using *pixels* - short for picture *elements* - which are confined to a rectangular grid. Such jaggies are usually the most visible with near-horizontal or near-vertical lines.
- "Plate 4" shows a *flat-shaded, unlit* version of the scene. The objects in the scene are now shown as solid. They appear "flat" in the sense that only one color is used to render each polygon, so they don't appear smoothly rounded. There are no effects from any light sources.
- "Plate 5" shows a *lit, smooth-shaded* version of the scene. Note how the scene looks much more realistic and three-dimensional when the objects are shaded to respond to the light sources in the room as if the objects were smoothly rounded.
- "Plate 6" adds *shadows* and *textures* to the previous version of the scene. Shadows aren't an explicitly defined feature of OpenGL (there is no "shadow command"). *Texture mapping* allows you to apply a two-dimensional image onto a three-dimensional object. In this scene, the top on the table surface is the most vibrant example of texture mapping.

- "Plate 7" shows a *motion-blurred* object in the scene. The sphinx (or dog, depending on your Rorschach tendencies) appears to be captured moving forward, leaving a blurred trace of its path of motion.
- "Plate 8" shows the scene as it's drawn for the cover of the book from a different viewpoint. This plate illustrates that the image really is a snapshot of models of three-dimensional objects.
- "Plate 9" brings back the use of fog, which was seen in "Plate 2," to show the presence of smoke particles in the air. Note how the same effect in "Plate 2" now has a more dramatic impact in "Plate 9."
- "Plate 10" shows the *depth-of-field effect*, which simulates the inability of a camera lens to maintain all objects in a photographed scene in focus. The camera focuses on a particular spot in the scene. Objects that are significantly closer or farther than that spot are somewhat blurred.

The color plates give you an idea of the kinds of things you can do with the OpenGL graphics system. The following list briefly describes the major graphics operations which OpenGL performs to render an image on the screen.

Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.) Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene.

Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pasting a texture onto the objects, or some combination of these three actions.

Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called *rasterization.*

During these stages, OpenGL might perform other operations, such as eliminating parts of objects that are hidden by other objects. In addition, after the scene is rasterized but before it's drawn on the screen, you can perform some operations on the pixel data if you want.

In some implementations (such as with the X Window System), OpenGL is designed to work even if the computer that displays the graphics you create isn't the computer that runs your graphics program. This might be the case if you work in a networked computer environment where many computers are connected to one another by a digital network. In this situation, the computer on which your program runs and issues OpenGL drawing commands is called the client, and the computer that receives those commands and performs the drawing is called the server. The format for transmitting OpenGL commands (called the *protocol*) from the client to the server is always the same, so OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server.

2.3.2 A Smidgen of OpenGL Code [3]

OpenGL program can be complicated. However, the basic structure of a useful program can be simple: Its tasks are to initialize certain states that control how OpenGL renders and to specify objects to be rendered.

Rendering is the process by which a computer creates images from models. These *models*, or objects, are constructed from geometric primitives points, lines, and polygons that are specified by their vertices.

The final rendered image consists of pixels drawn on the screen; a pixel is the smallest visible element the display hardware can put on the screen. Information about the pixels (for instance, what color they're supposed to be) is organized in memory into bitplanes.

A bitplane is an area of memory that holds one bit of information for every pixel on the screen; the bit might indicate how red a particular pixel is supposed to be, for example. The bitplanes are themselves organized into a *framebuffer*, which holds all the information that the graphics display needs to control the color and intensity of all the pixels on the screen.

Now look at what an OpenGL program might look like. Example renders a white rectangle on a black background, as shown in Figure 2.2.

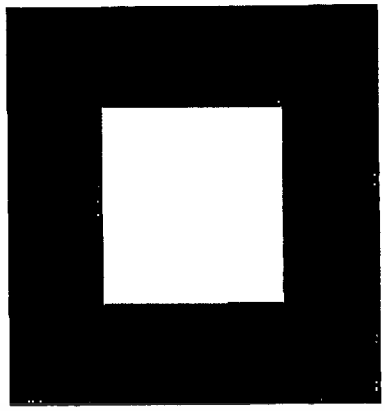


Figure 2.2 White Rectangles on a Black Background

Example: Chunk of OpenGL Code

```
#include <whateverYouNeed.h>
main() {

InitializeAWindowPlease();

glClearColor (0.0, 0.0, 0.0, 0.0);
glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
glBegin(GL_POLYGON);
    glVertex3f (0.25, 0.25, 0.0);
    glVertex3f (0.75, 0.25, 0.0);
    glVertex3f (0.75, 0.75, 0.0);
    glVertex3f (0.25, 0.75, 0.0);
glEnd();
glFlush();

UpdateTheWindowAndCheckForEvents();
}
```

The first line of the main() routine initializes a *window* on the screen: The InitializeAWindowPlease() routine is meant as a placeholder for window system-specific routines, which are generally not OpenGL calls. The next two lines are OpenGL commands that clear the window to black: glClearColor() establishes what color the window will be cleared to, and glClear() actually clears the window. Once the clearing color is set, the window is cleared to that color whenever glClear() is called. This clearing color can be changed with another call to glClearColor(). Similarly, the glColor3f() command establishes what color to use for drawing objects - in this case, the color is white. All objects drawn after these points use this color, until it's changed with another call to set the color.

The next OpenGL command used in the program, glOrtho(), specifies the coordinate system OpenGL assumes as it draws the final image and how the image gets mapped to the screen. The next calls, which are bracketed by glBegin() and glEnd(), define the object to be drawn - in this example, a polygon with four vertices. The polygon's "corners" are defined by the glVertex3f() commands. As you might be able to guess from the arguments, which are (*x*, *y*, *z*) coordinates, the polygon is a rectangle on the z=0 plane.

Finally, glFlush() ensures that the drawing commands are actually executed rather than stored in a *buffer* awaiting additional OpenGL commands.

The UpdateTheWindowAndCheckForEvents() placeholder routine manages the contents of the window and begins event processing.

Actually, this piece of OpenGL code isn't well structured. You may be asking, "What happens if I try to move or resize the window?" Or, "Do I need to reset the coordinate system each time I draw the rectangle?" Later in this chapter, you will see replacements for both InitializeAWindowPlease() and UpdateTheWindowAndCheckForEvents() that actually work but will require restructuring the code to make it efficient.

2.3.3 OpenGL Command Syntax [3]

OpenGL commands use the prefix gl and initial capital letters for each word making up the command name (recall glClearColor(), for example). Similarly, OpenGL defined constants begin with GL_, use all capital letters, and use underscores to separate words (like

GL_COLOR_BUFFER_BIT). Users might also have noticed some seemingly extraneous letters appended to some command names (for example, the 3f in glColor3f() and glVertex3f()). It's true that the Color part of the command name glColor3f() is enough to define the command as one that sets the current color. However, more than one such command has been defined so that you can use different types of arguments. In particular, the 3 part of the suffix indicates that three arguments are given; another version of the Color command takes four arguments. The f part of the suffix indicates that the arguments are floating-point numbers. Having different formats allows OpenGL to accept the user's data in his or her own data format.

Some OpenGL commands accept as many as 8 different data types for their arguments. The letters used as suffixes to specify these data types for ISO C implementations of OpenGL are shown in Table 2.1, along with the corresponding OpenGL type definitions. The particular implementation of OpenGL that you are using might not follow this scheme exactly; an implementation in C++ or Ada as shown in Table 2.1, for example, would not need to.

Table 2.1 Command Suffixes and Argument Data Types

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Thus, the two commands

```
glVertex2i(1, 3);
glVertex2f(1.0, 3.0);
```

are equivalent, except that the first specifies the vertex's coordinates as 32-bit integers, and the second specifies them as single-precision floating-point numbers.

Note: Implementations of OpenGL have leeway in selecting which C data type to use to represent OpenGL data types. If you resolutely use the OpenGL defined data types throughout your application, you will avoid mismatched types when porting your code between different implementations.

Some OpenGL commands can take a final letter v, which indicates that the command takes a pointer to a vector (or array) of values rather than a series of individual arguments. Many commands have both vector and nonvector versions, but some commands accept only individual arguments and others require that at least some of the arguments be specified as a vector. The following lines show how you might use a vector and a nonvector version of the command that sets the current color:

```
glColor3f(1.0, 0.0, 0.0);
GLfloatcolor_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);
```

Finally, OpenGL defines the typedefGLvoid. This is most often used for OpenGL commands that accept pointers to arrays of values.

OpenGL commands are referred to by their base names only, and an asterisk is included to indicate that there may be more to the command name. For example, glColor*() stands for all variations of the command you use to set the current color. If we want to make a specific point about one version of a particular command, we include the suffix necessary to define that version. For example, glVertex*v() refers to all the vector versions of the command you use to specify vertices.

2.3.4 Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which requires inclusion of the glu.h header file. So almost every OpenGL source file begins with;

```
#include <GL/gl.h>
#include <GL/glu.h>
```

If users are directly accessing a window interface library to support OpenGL, such as GLX, AGL, PGL, or WGL, their must include additional header files. For example, if users are calling GLX, there may need to add these lines to the code

```
#include <X11/Xlib.h>
#include <GL/glx.h>
```

If users are using GLUT for managing your window manager tasks, you should include;

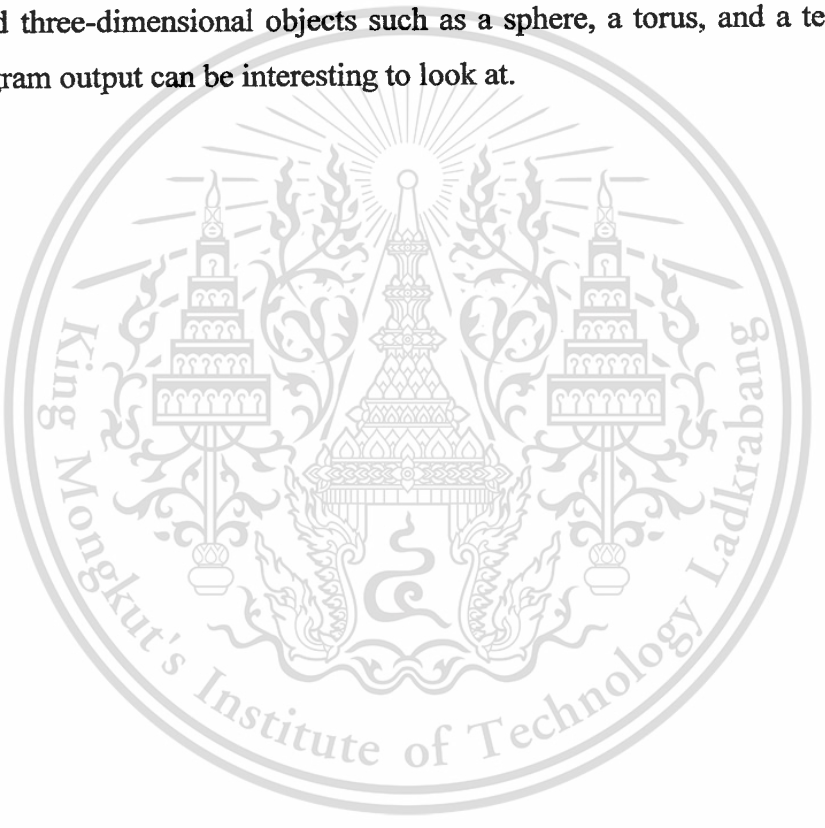
```
#include <GL/glut.h>
```

Note that glut.h includes gl.h, glu.h, and glx.h automatically, so including all three files is redundant. GLUT for Microsoft Windows includes the appropriate header file to access WGL.

2.3.5 GLUT, the OpenGL Utility Toolkit

OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Unfortunately, it's impossible to write a complete graphics program without at least opening a window, and most interesting programs require a bit of users input or other services from the operating system or window system.

In addition, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. This way, snapshots of program output can be interesting to look at.



Chapter 3

System Design

This chapter will cover the design in the system. The designs are activity diagram, use case diagram, control hierarchy diagram, and users interface. The chapter ends by explaining how system works.

3.1 Use Case Diagram

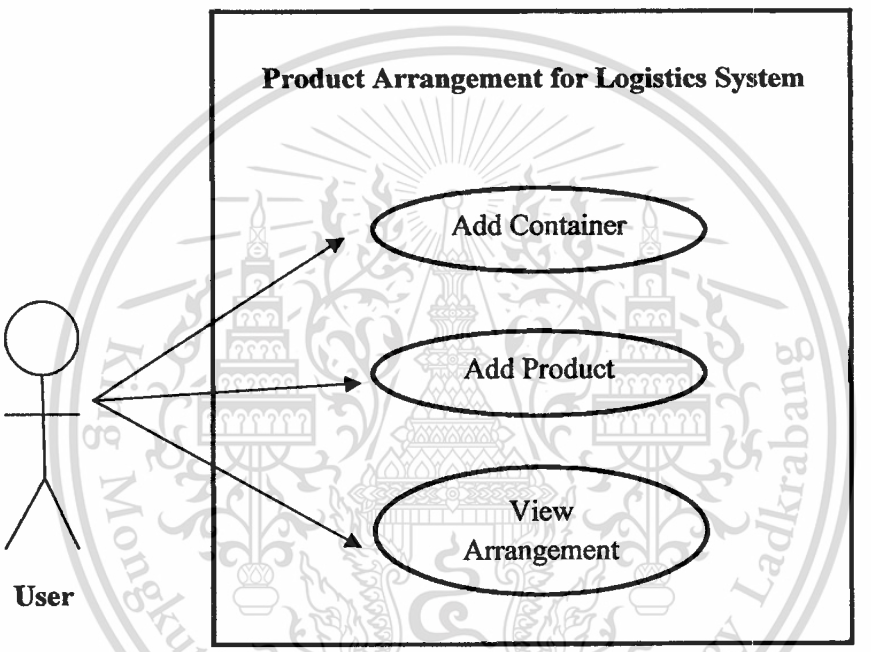


Figure 3.1 Use case diagram for Product arrangement project

This diagram shows that the users can add the container to store the product and add product to be arranged into the container. It also shows that the users can edit the information of the container or the product that s/he added into the program. Then, the users can view the arrangement result.

In the program, there is the container information for users to choose. After the users chose the container, there will be the boxes information display on the screen. The users will see menu to select the size first then the quantity later. Then the users can select the size and quantity again until they satisfy.

3.2.2 Activity Diagram of View Arrangement Result

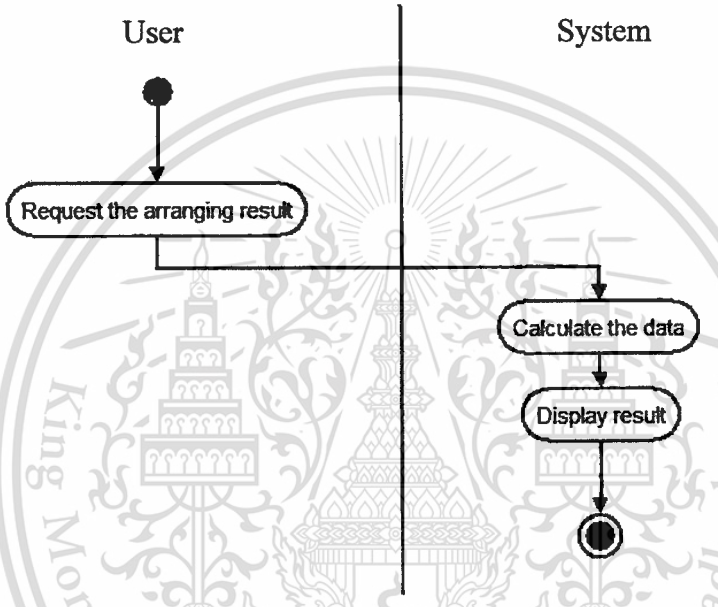


Figure 3.3 Activity diagram of View Arrangement Result

When the users enter 0 in the program to start calculation, the program will calculate the data and display the result of the arrangement.

3.3 Control Hierarchy Diagram

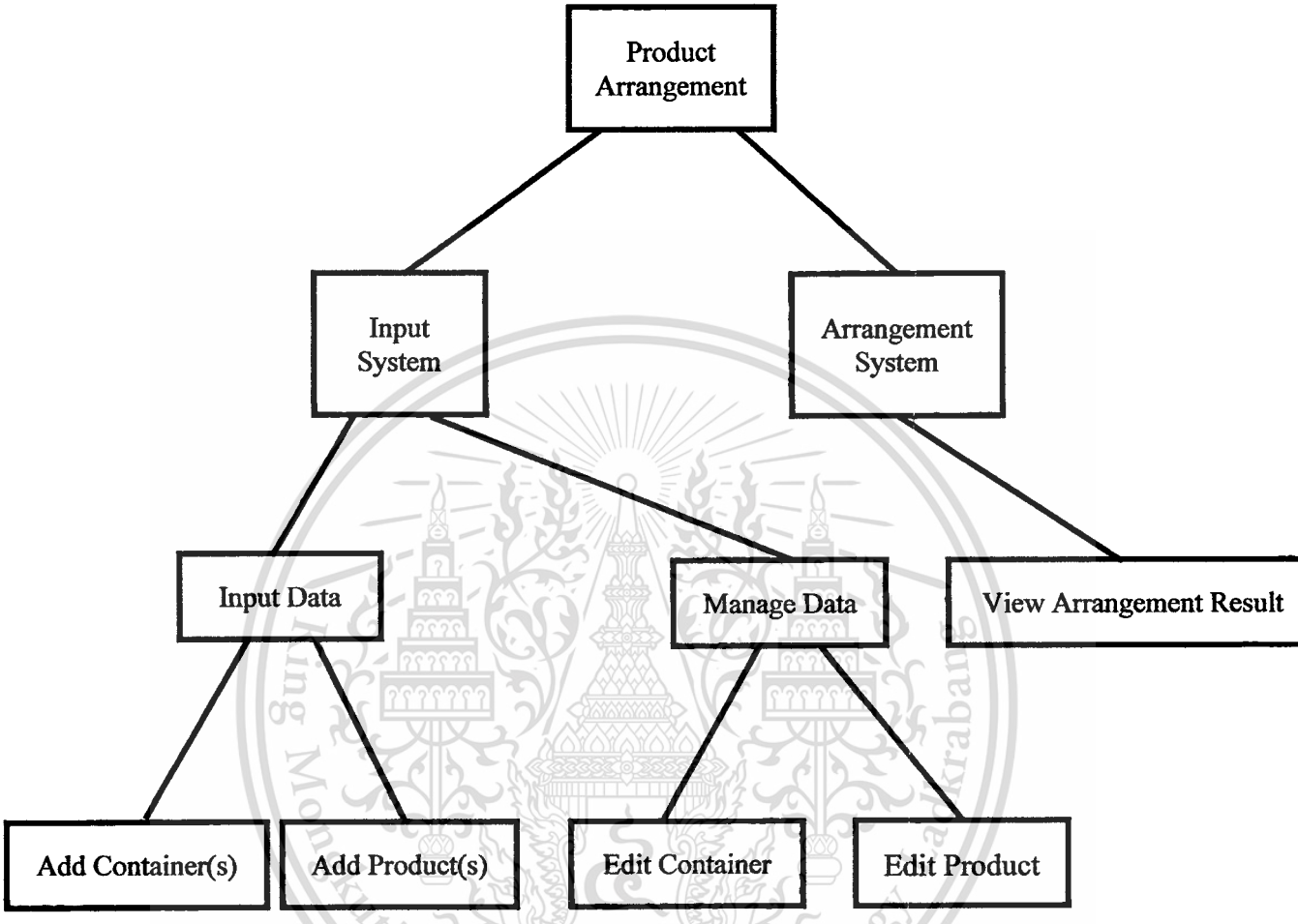


Figure 3.4 Control hierarchy diagram

Control hierarchy diagram shows the structure of the program step by step. From the Figure 3.4, the program consists of 2 parts; Input System section and Arrangement System section. Input System section divided into Input Data and Manage Data. The data includes; container and products. For Arrangement System section, the users can view the arrangement result.

3.4 Graphic User Interface

3.4.1 GUI in the 1st Page of the Program (Main Page)

There are 5 types of boxes to be loaded into the vehicles as follows;

Table 3.1 Box information

Type	Size	Color
1.Small Box	31.12cm. x 27.69cm. x 3.81cm.	
2.Medium Box	33.66cm. x 29.21cm. x 6.03cm.	
3.Large Box	45.40cm. x 31.43cm. x 7.62cm.	
4.Extra Large Box	40.16cm. x 32.86cm. x 25.88cm.	
5.Double Extra Large Box	54.80cm. x 42.10cm. x 33.50cm.	

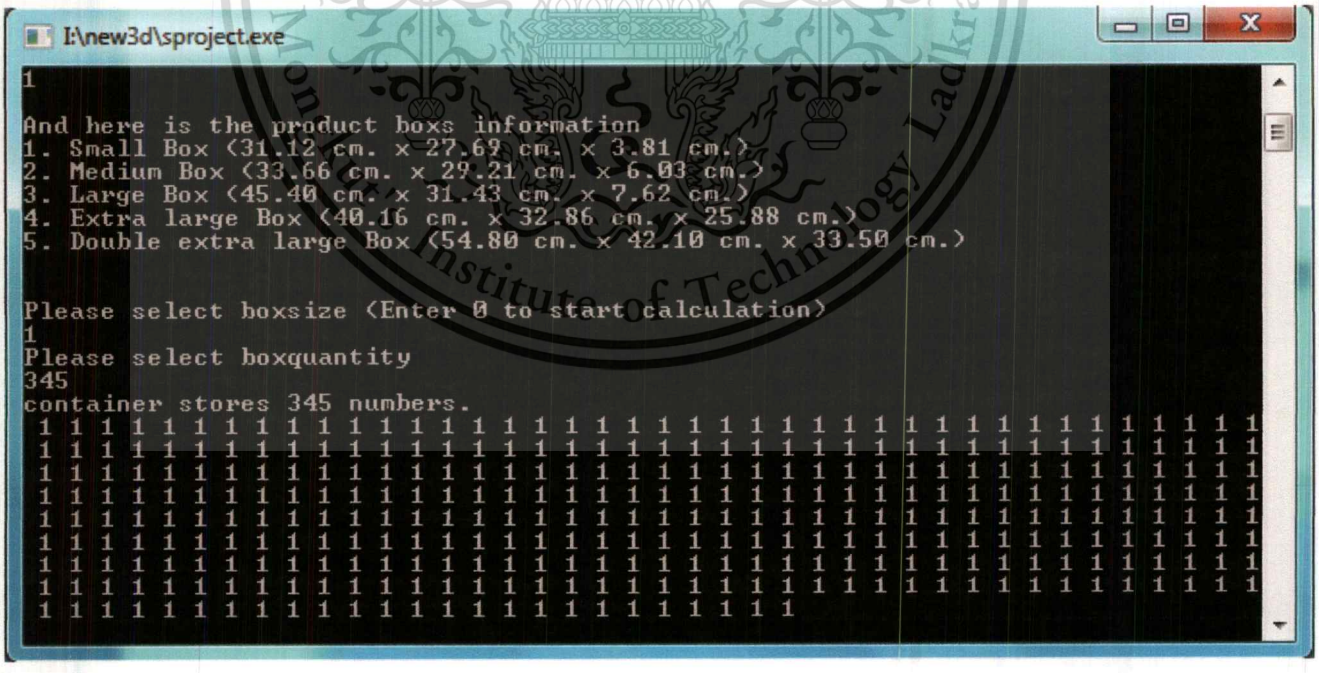


Figure 3.5 Main page

3.4.2 Result page

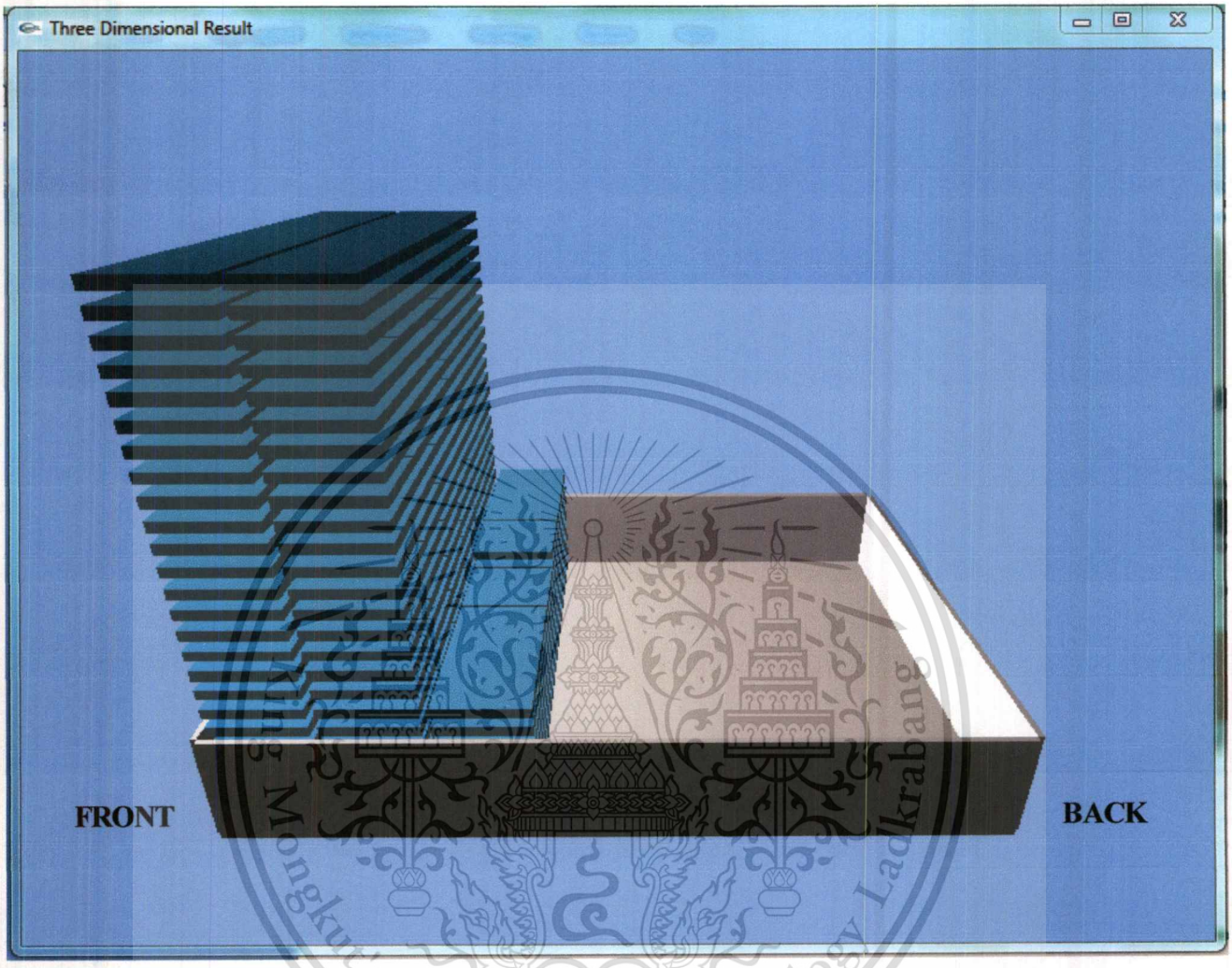


Figure 3.6 The example of result page

Chapter 4

Experimental Results

4.1 The Experimental Results

In the experimental results, many cases of algorithms and user's requirements were tested. The tested algorithms were First-Fit, First-Fit Decreasing, and Last-Fit.

4.1.1 Case 1

In this case, suppose that the user has the packages to be arranged as shown in Table 4.1. If the user entered the information in the following list, the sorting part of the calculation will be different in each method.

Table 4.1 User's requirement in case1

Box Size	Quantity (boxes)
1. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.)	100
2. Double extra large box size (54.80 cm. x 42.10 cm. x 33.50 cm.)	60
3. Small box size (31.12 cm. x 27.69 cm. x 3.81 cm.)	200

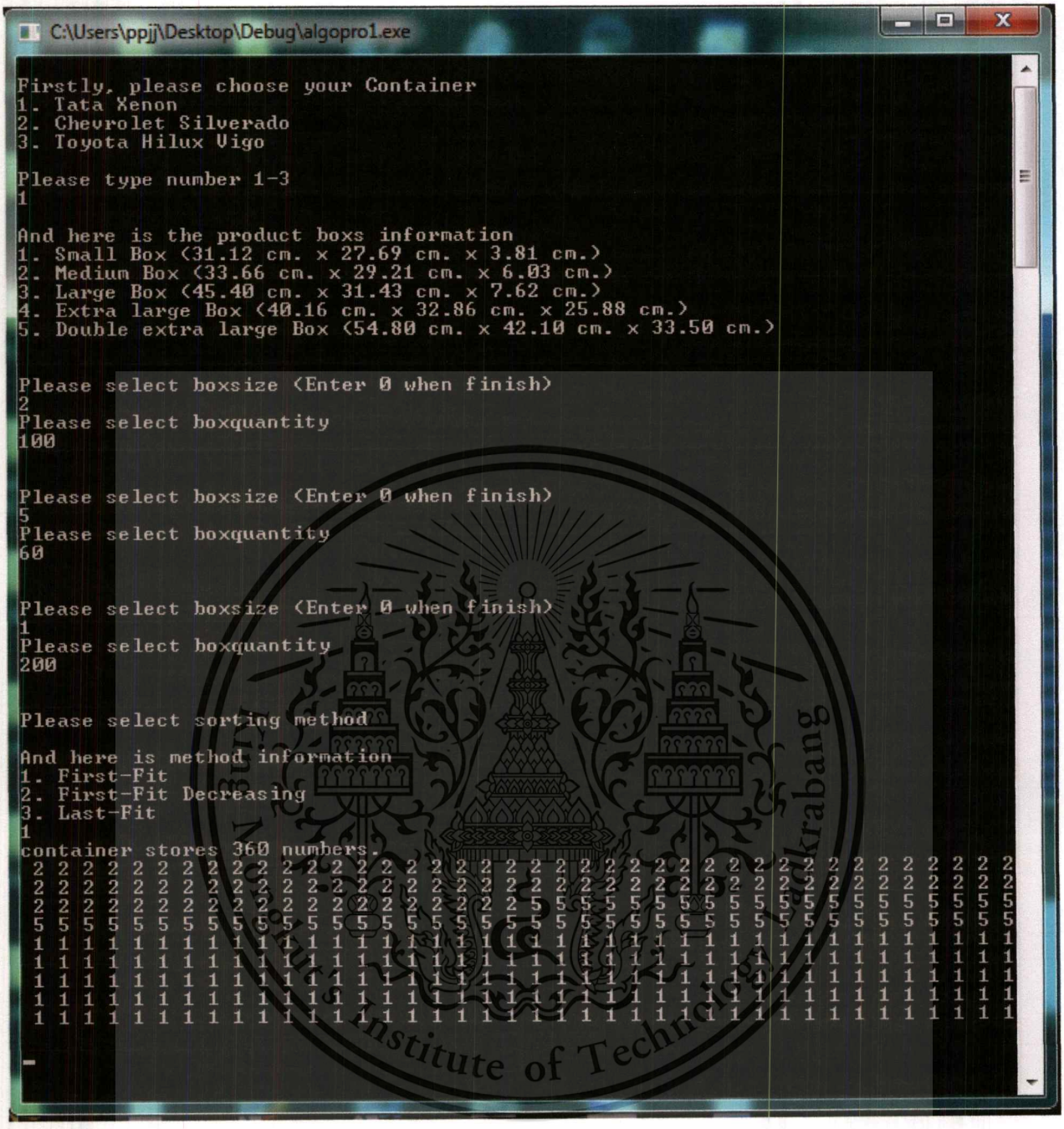


Figure 4.1 The result of arrangement in case 1

Figure 4.1 shows the input example for the First-Fit sorting method. For selecting other methods, the user just has to change the method information which are "1" as First-Fit, "2" as First-Fit Decreasing, and "3" as Last-Fit.

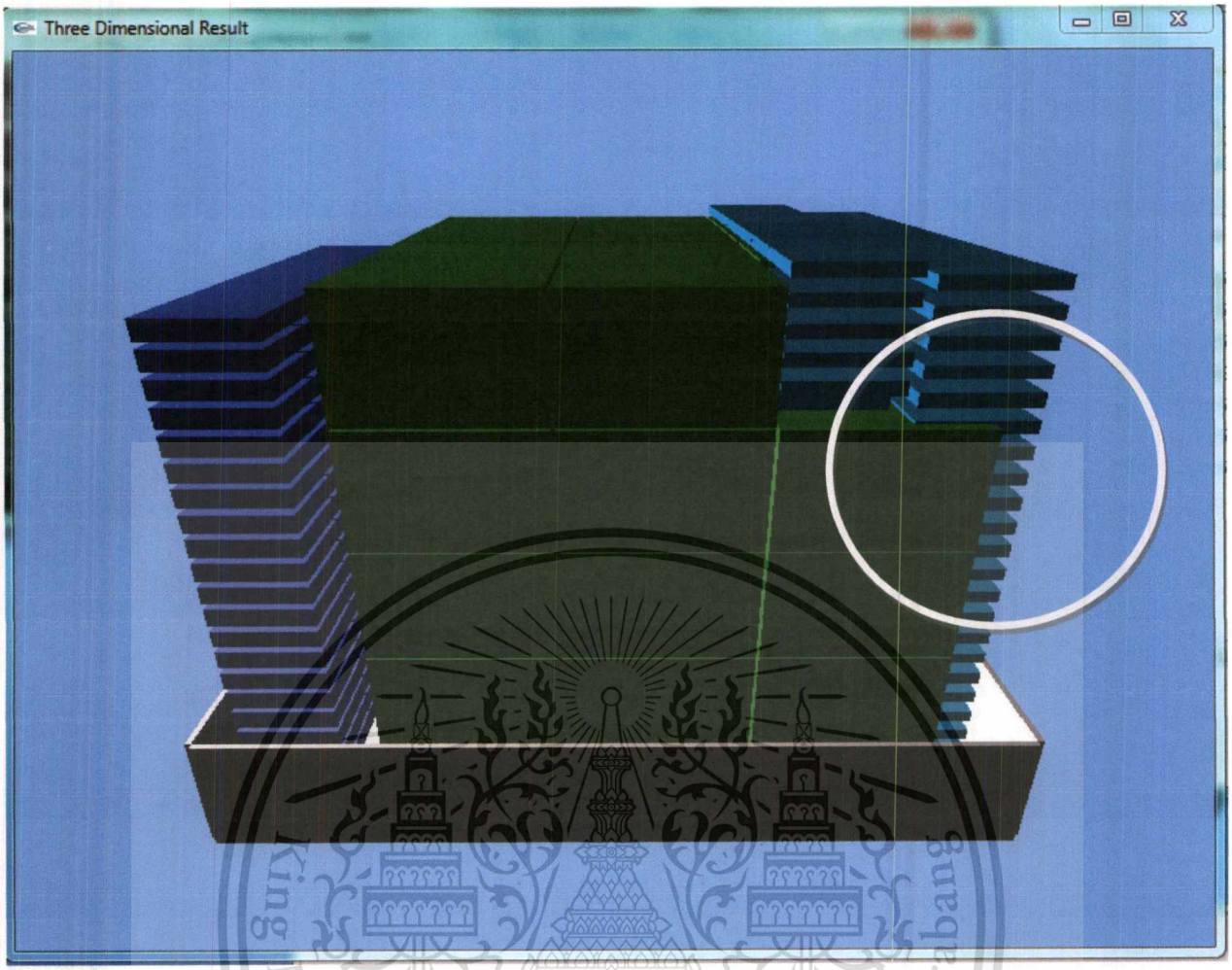


Figure 4.2 The graphical result of the First-Fit algorithm of case1

Figure 4.2 shows the result of the First-Fit sorting method in case 1. The figure indicated that there was an unsorted space on the right side of the Medium box and the items overlapped at the right side of the container.

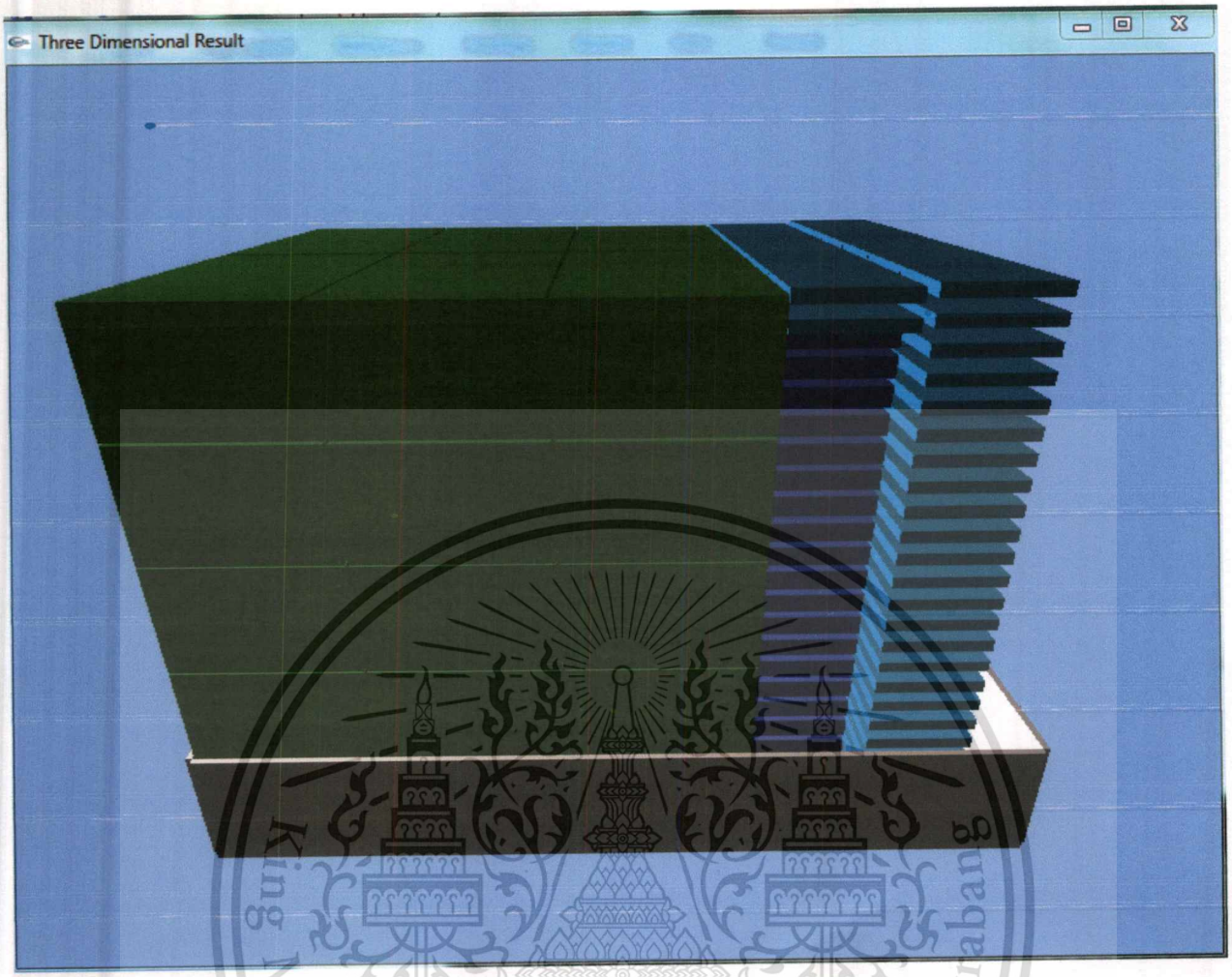


Figure 4.3 The graphical result of the First-Fit Decreasing algorithm of case1

Figure 4.3 shows the result of the First-Fit Decreasing sorting method in case 1. Everything fit perfectly in this case with no space left for inserting more boxes.

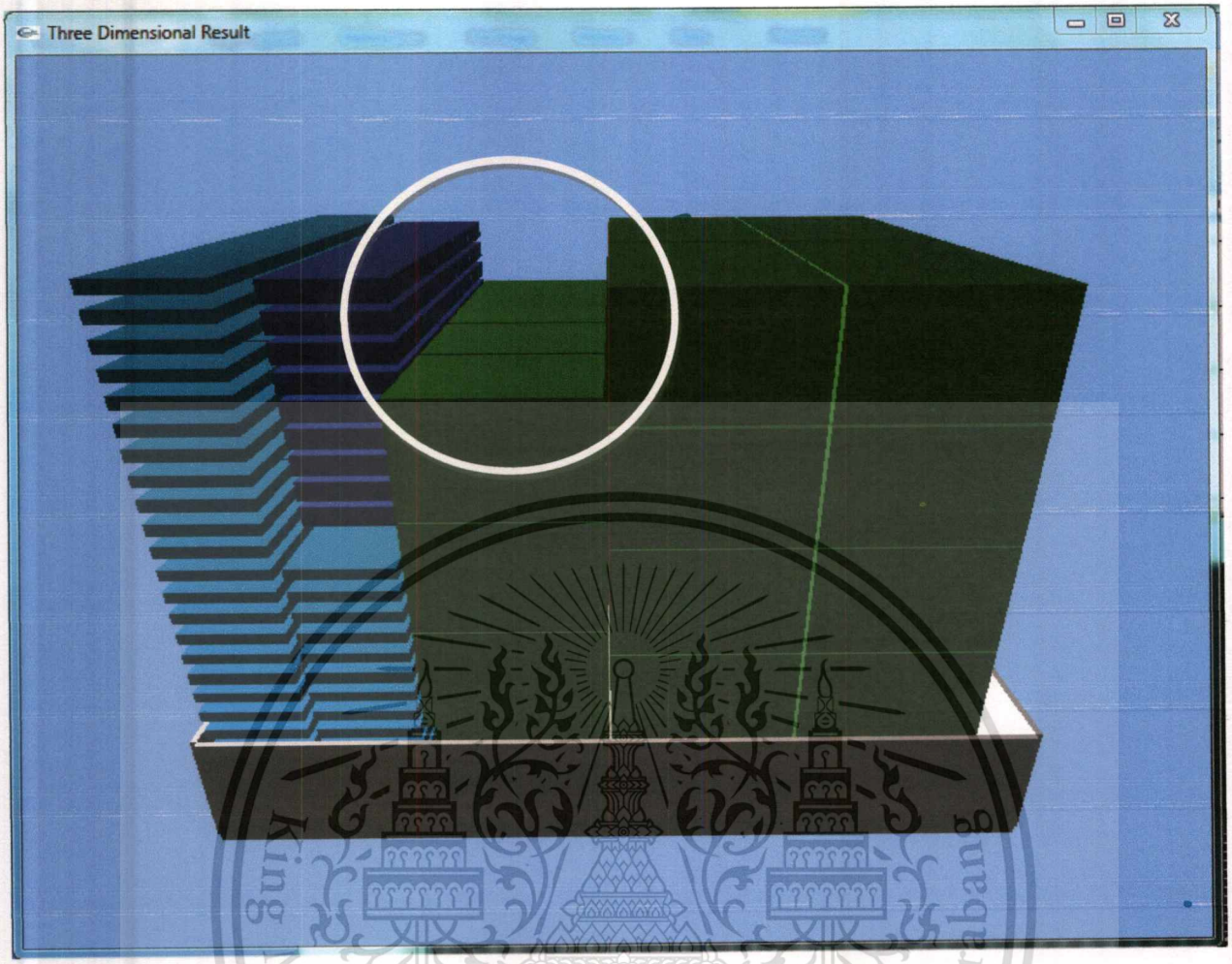


Figure 4.4 The graphical result of the Last-Fit algorithm of case1

Figure 4.4 shows the result of the Last-Fit sorting method in case 1. According to the result of the First-Fit Decreasing algorithm, which is the most complete one in this program, shows that there should be no space left. However, in this result of using the Last-Fit algorithm, there were some spaces left on top of the first row of the Double extra large boxes. So it might be concluded that the overlap occurred in this algorithm.

4.1.2 Case 2

In this case, suppose that the user has the packages to be arranged as shown in Table 4.2. If the user entered the information in the following list, the sorting part of the calculation will be different in each method.

Table 4.2 User's requirement in case 2

Box Size	Quantity (boxes)
1. Double extra large box size (54.80 cm. x 42.10 cm. x 33.50 cm.)	40
2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.)	20
3. Extra large box size (40.16 cm. x 32.86 cm. x 25.88 cm.)	10

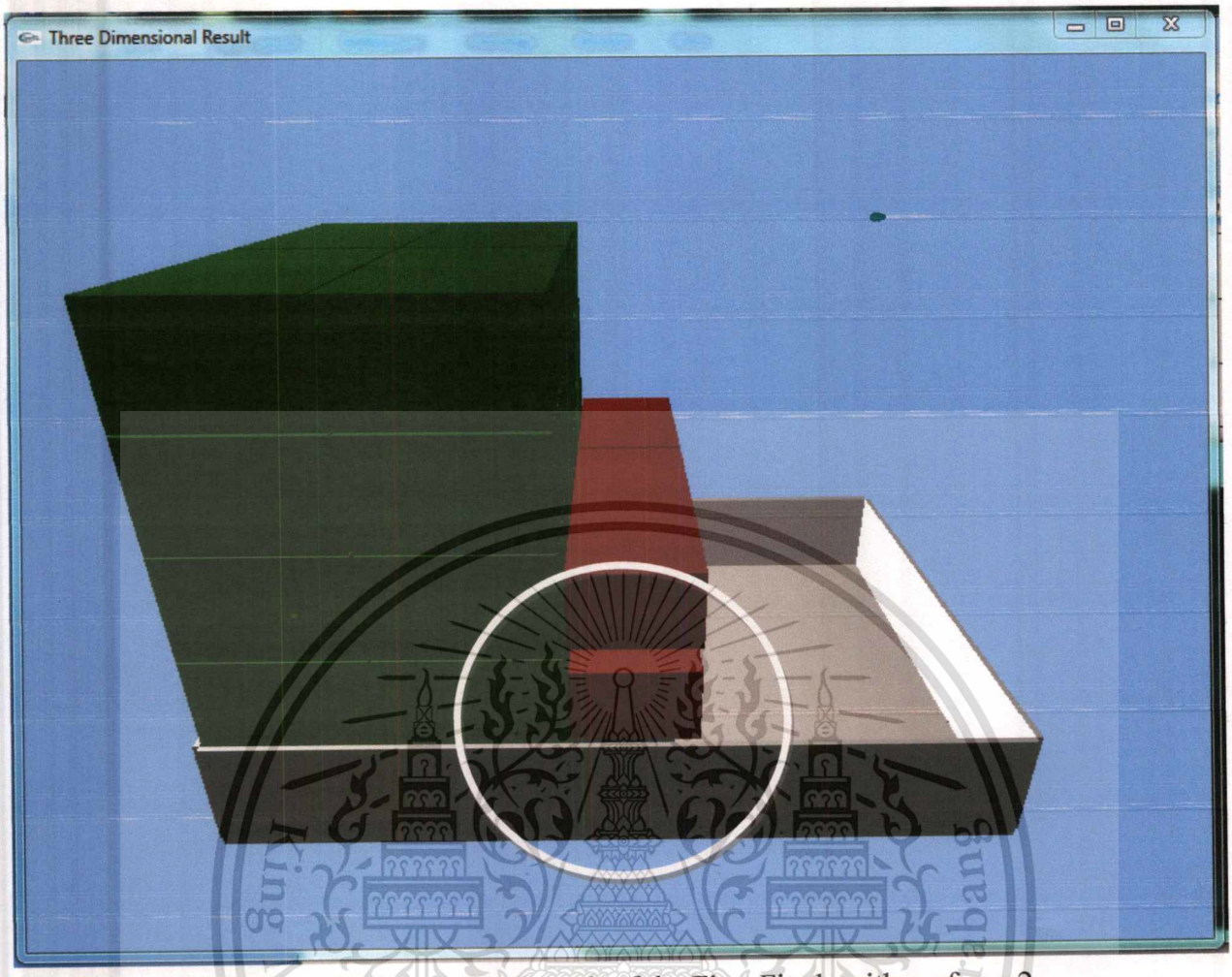


Figure 4.5 The graphical result of the First-Fit algorithm of case2

Figure 4.5 shows the result of the First-Fit sorting method in case 2. In this figure, the Medium boxes disappeared. So it might be concluded that the overlap occurred in this algorithm.

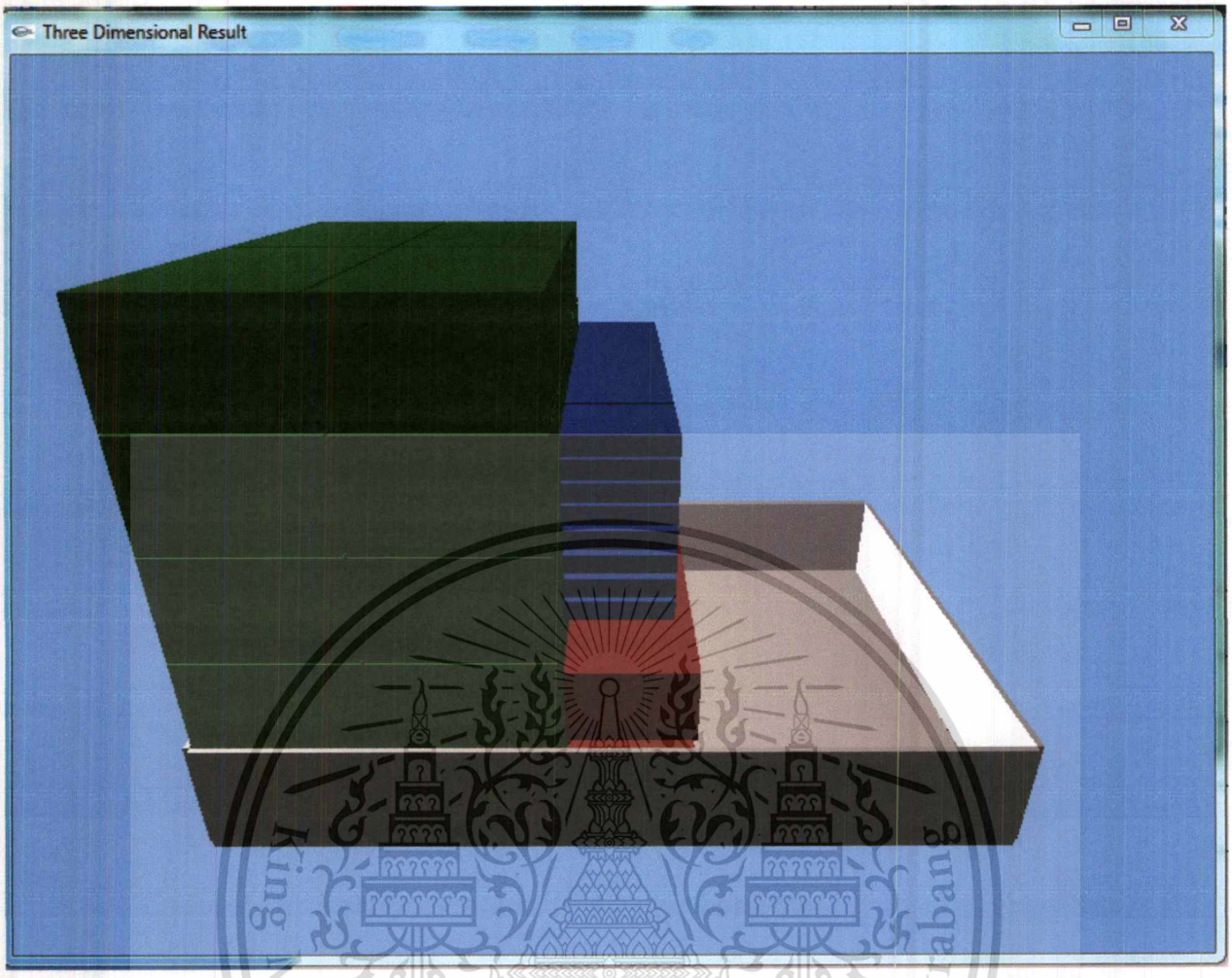


Figure 4.6 The graphical result of the First-Fit Decreasing algorithm of case2

Figure 4.6 shows the result of the First-Fit Decreasing sorting method in case 2. The items fit perfectly in this case.

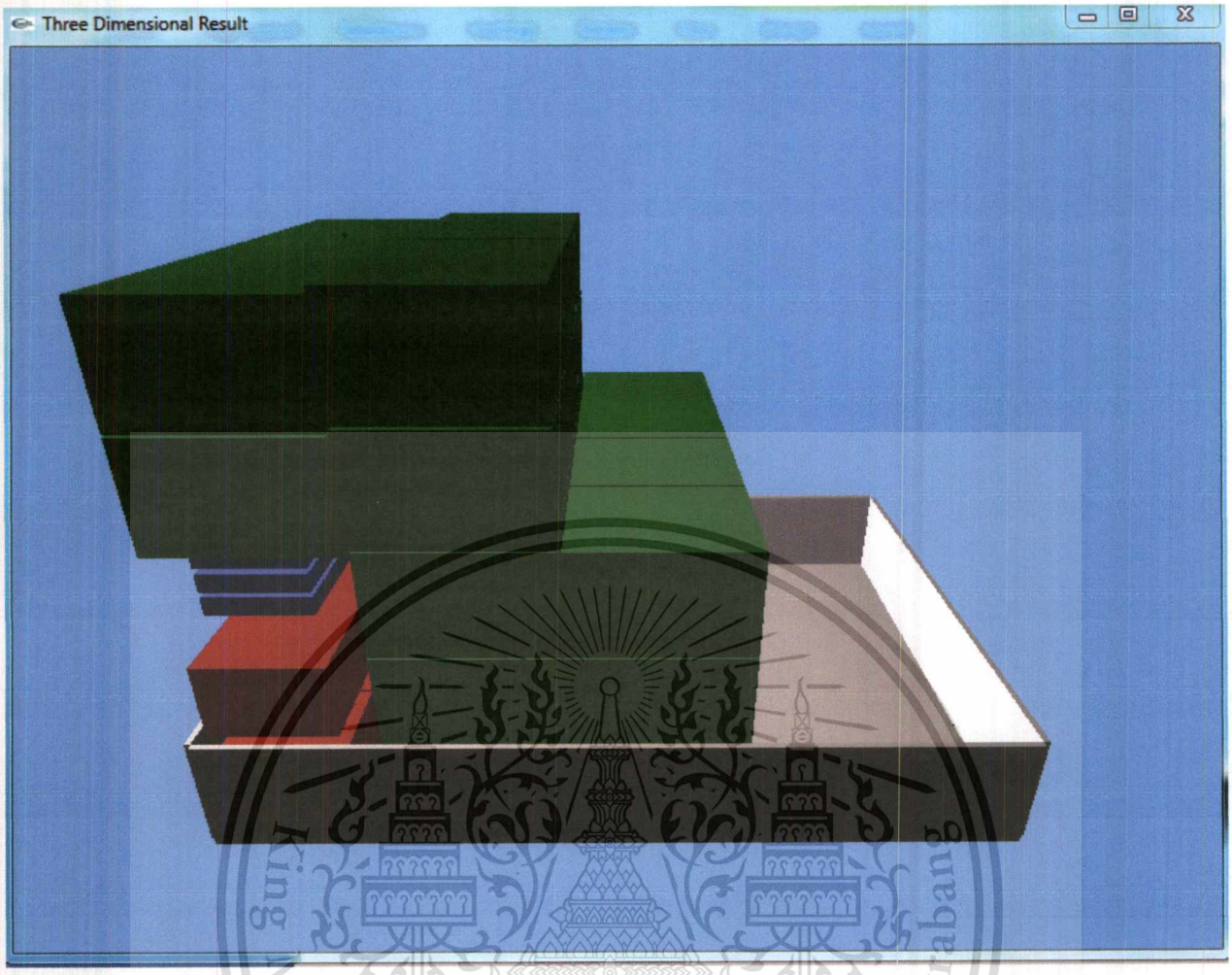


Figure 4.7 The graphical result of the Last-Fit algorithm of case2

Figure 4.7 shows the result of the Last-Fit sorting method in case 2. We can see that the space left in the long side of the container is less than what is left in the First-Fit Decreasing algorithm. There were some spaces left in the long side of the container less than in First-Fit Decreasing algorithm.

4.1.3 Case 3

In this case, suppose that the user has the packages to be arranged as shown in Table 4.3. If the user entered the information in the following list, the sorting part of the calculation will be different in each method.

Table 4.3 User's requirement in case 3

Box Size	Quantity (boxes)
1. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.)	40
2. Small box size (31.12 cm. x 27.69 cm. x 3.81 cm.)	50
3. Double extra large box size (54.80 cm. x 42.10 cm. x 33.50 cm.)	20

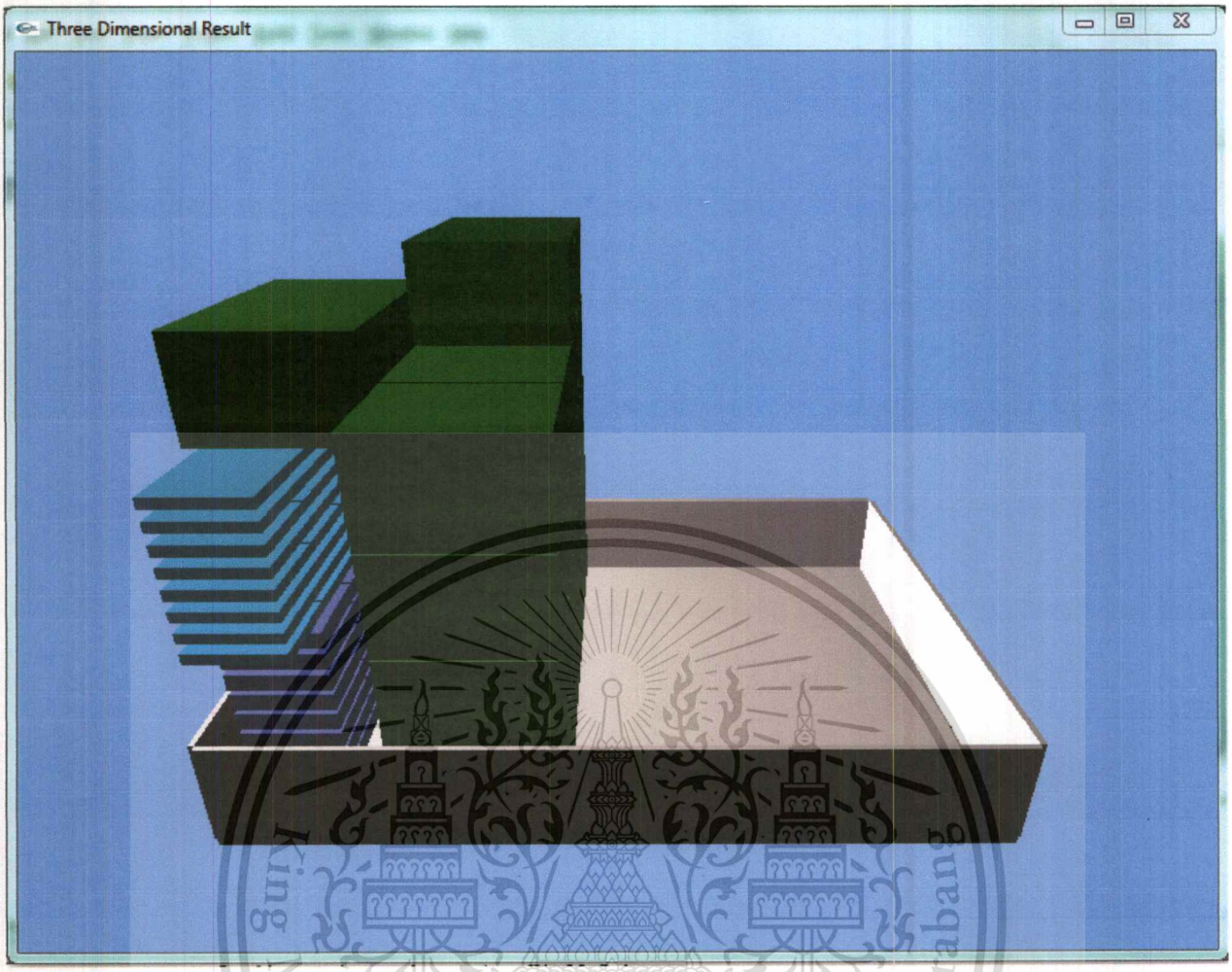


Figure 4.8 The graphical result of the First-Fit algorithm of case3

Figure 4.8 shows the result of the First-Fit sorting method in case 3. There were smaller spaces left in the long side of the container compared to other algorithms.

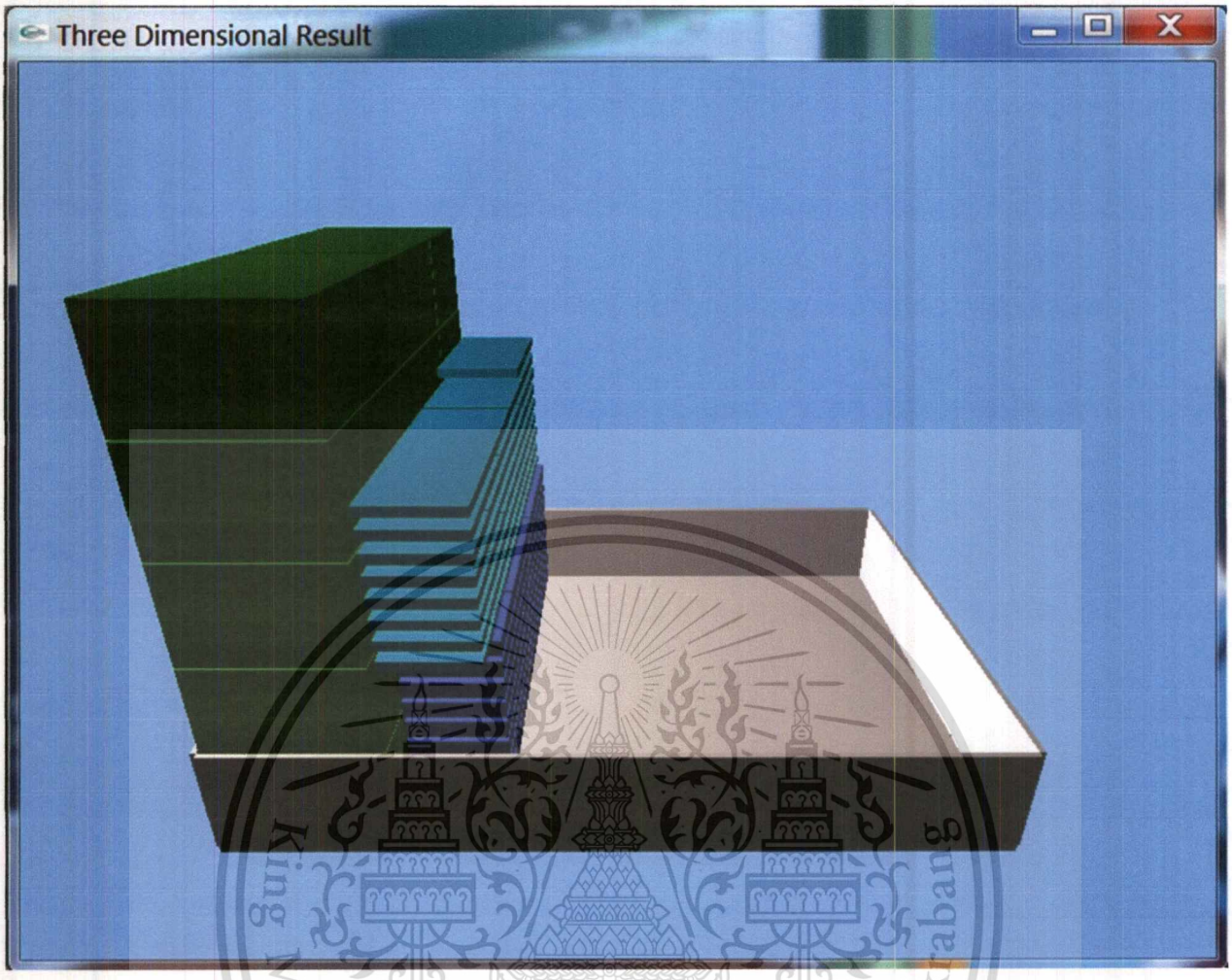


Figure 4.9 The graphical result of the First-Fit Decreasing algorithm of case3

Figure 4.9 shows the result of the First-Fit Decreasing sorting method in case 3. There were larger spaces left on top of the second row than the Last-Fit algorithm.

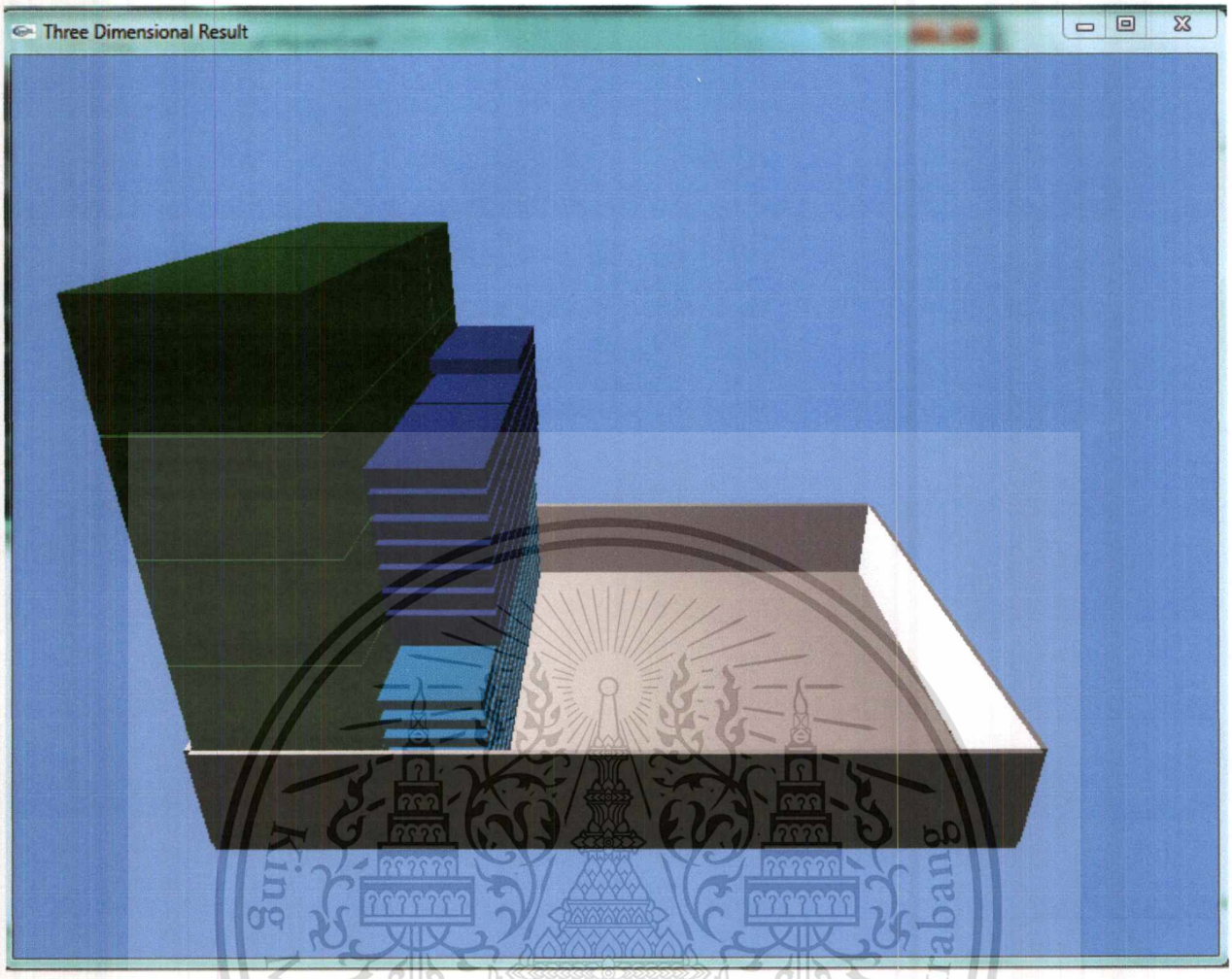


Figure 4.10 The graphical result of the Last-Fit algorithm of case3

Figure 4.10 shows the result of the Last-Fit sorting method in case 3.

4.1.4 Case 4

In this case, suppose that the user has the packages to be arranged as shown in Table 4.4. If the user entered the information in the following list, the sorting part of the calculation will be different in each method.

Table 4.4 User's requirement in case 4

Box Size	Quantity (boxes)
1. Small box size (31.12 cm. x 27.69 cm. x 3.81 cm.)	100
2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.)	100

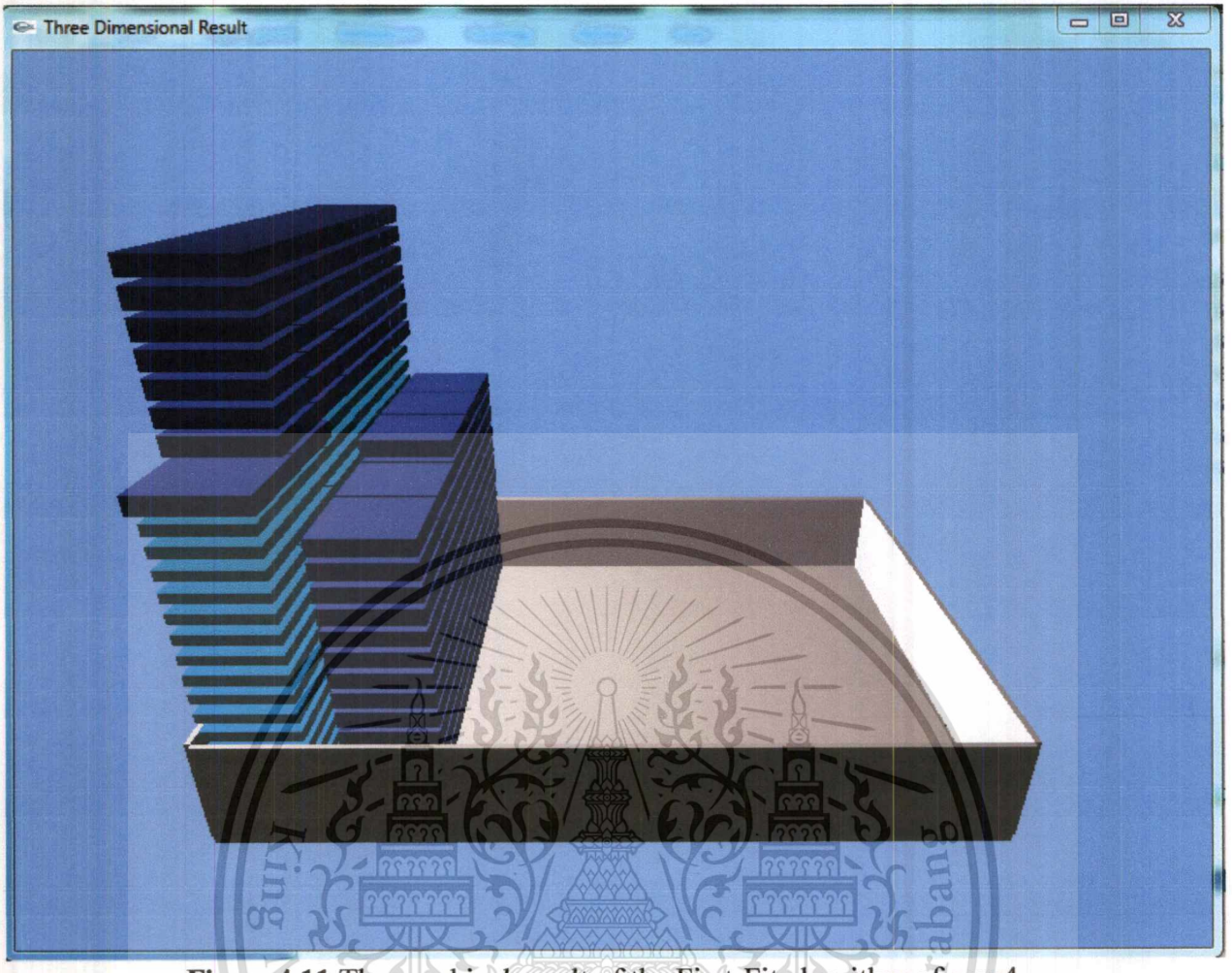


Figure 4.11 The graphical result of the First-Fit algorithm of case4

Figure 4.11 shows the result of the First-Fit sorting method in case 4. There were smaller spaces left in the long side of the container compared to other algorithms.

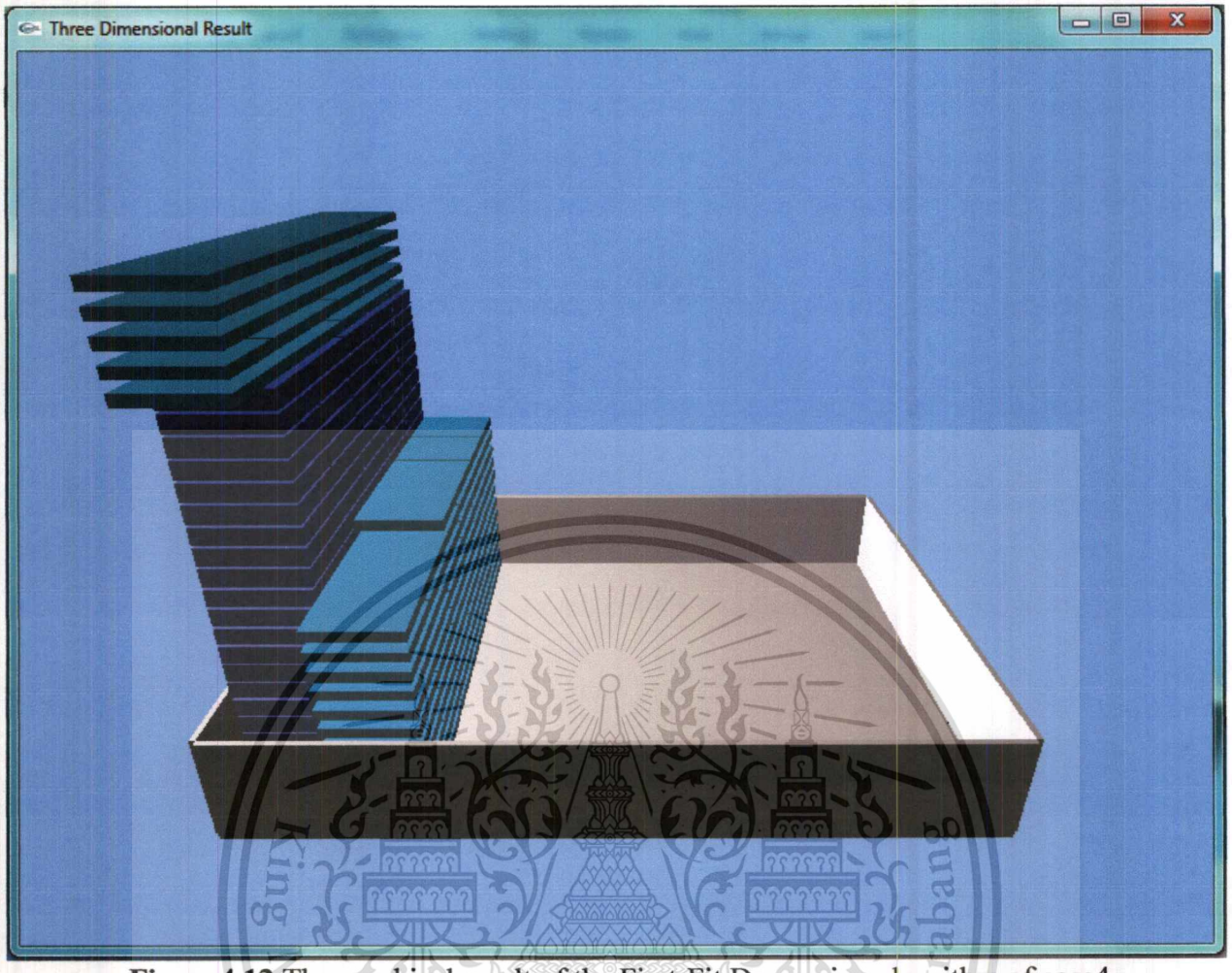


Figure 4.12 The graphical result of the First-Fit Decreasing algorithm of case4

Figure 4.12 shows the result of the First-Fit Decreasing sorting method in case 4. There were smaller spaces left on top of the second row than the First Fit algorithm.

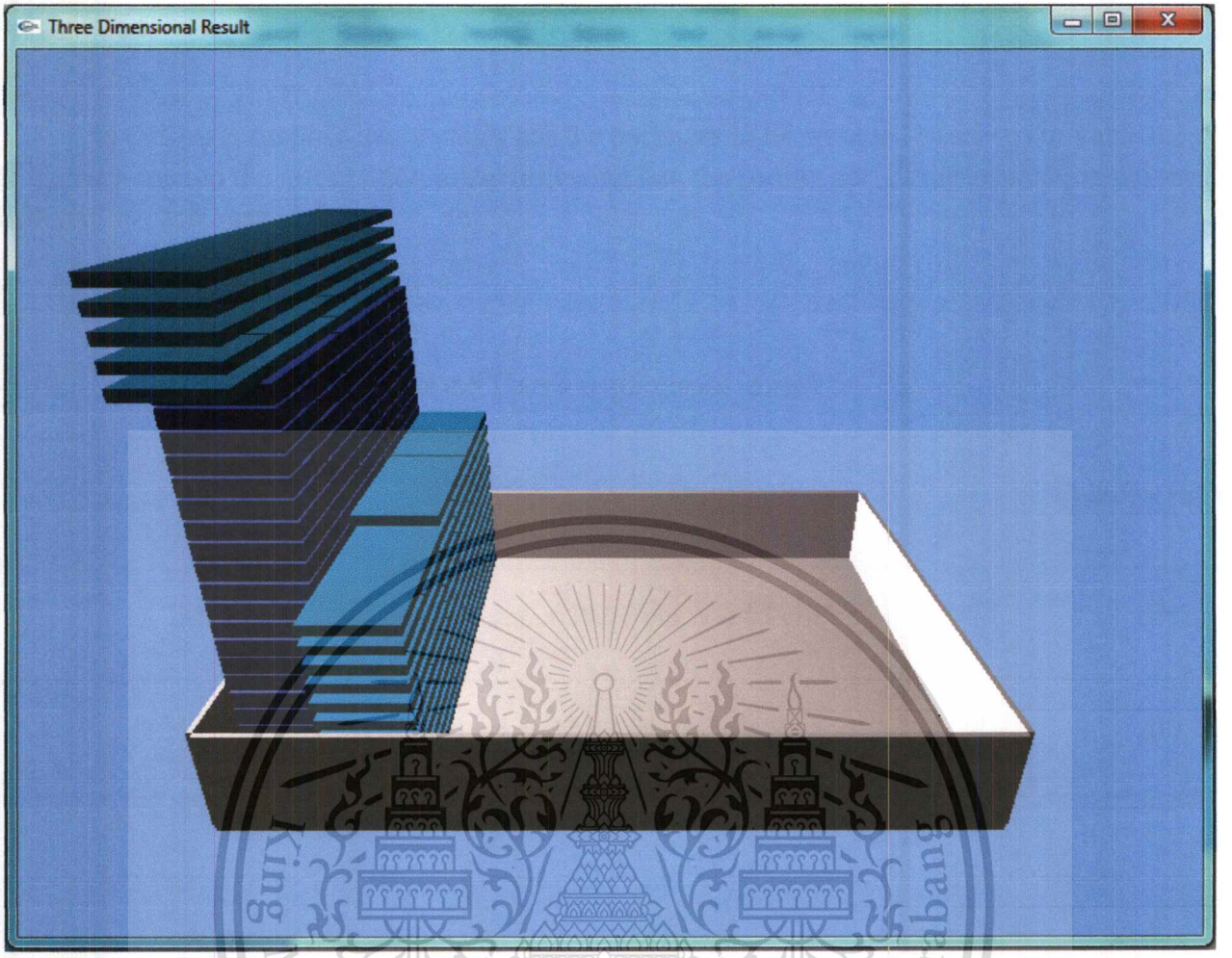


Figure 4.13 The graphical result of the Last-Fit algorithm of case4

Figure 4.13 shows the result of the Last-Fit sorting method in case 4.

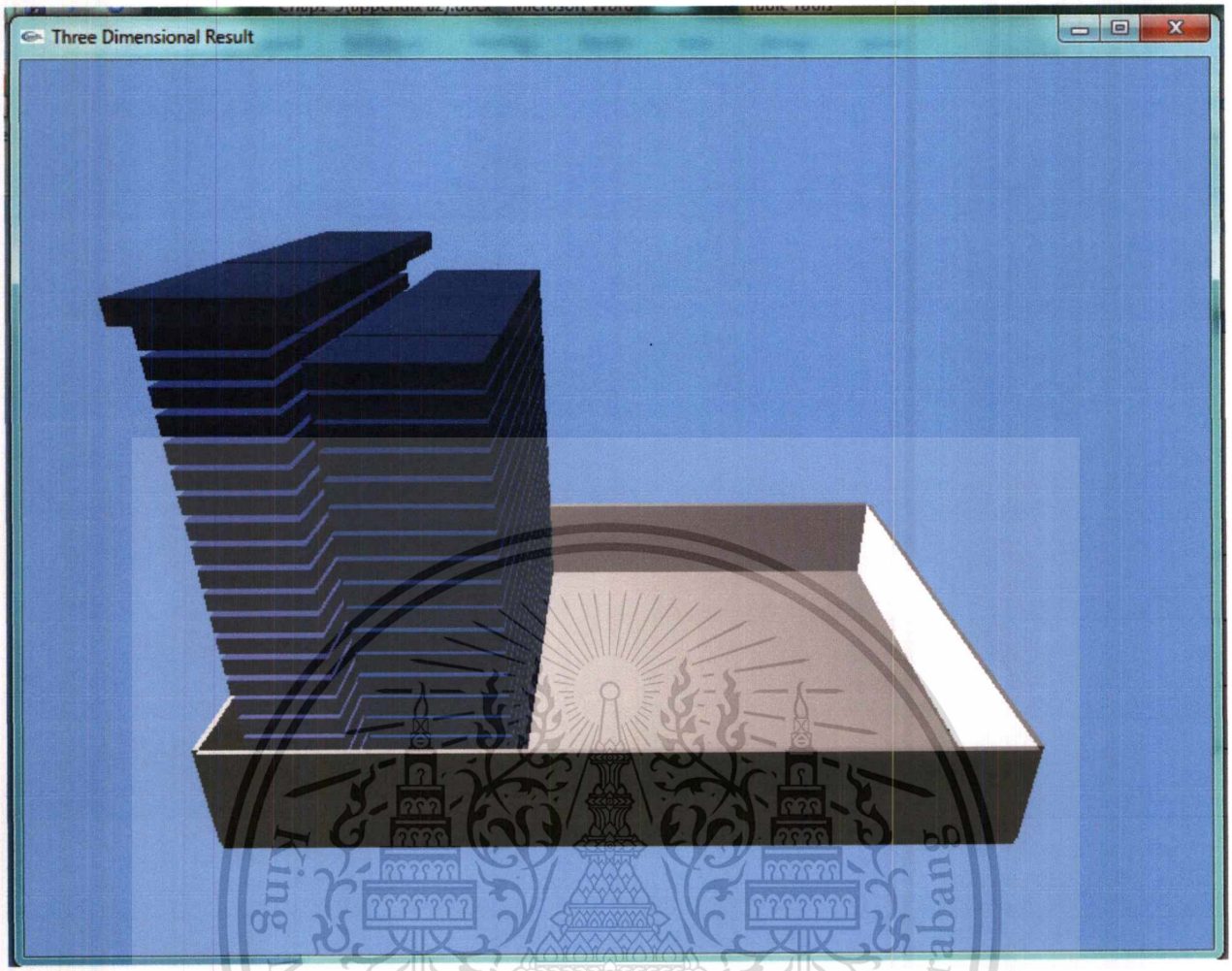


Figure 4.14 The graphical result of the First-Fit algorithm of case5

Figure 4.14 shows the result of the First-Fit sorting method in case 5. There were smaller spaces left in the long side of the container compared to other algorithms.

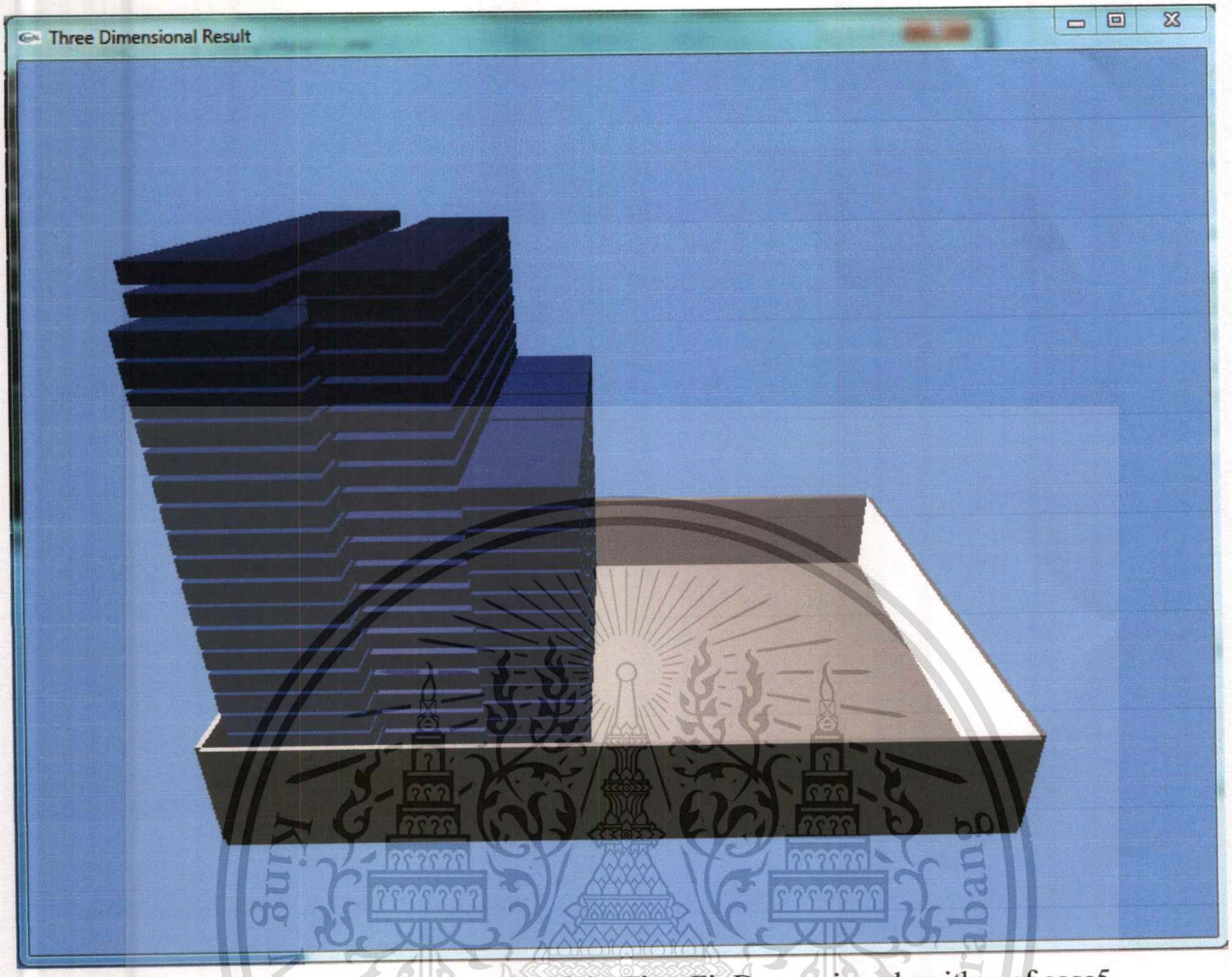


Figure 4.15 The graphical result of the First-Fit Decreasing algorithm of case5

Figure 4.15 shows the result of the First-Fit Decreasing sorting method in case 5.

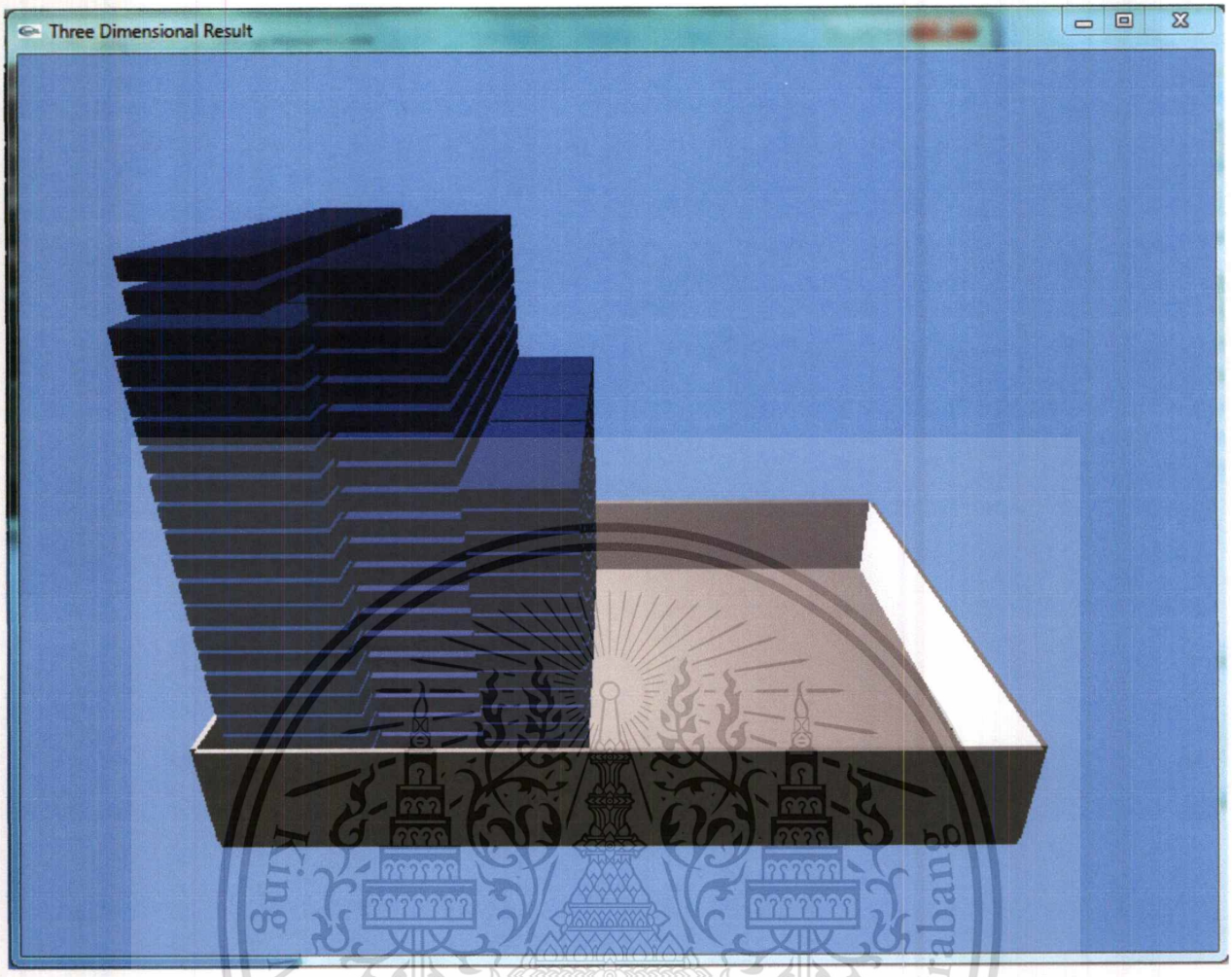


Figure 4.16 The graphical result of the Last-Fit algorithm of case5

Figure 4.16 shows the result of the Last-Fit sorting method in case 5.

4.1.6 Conclusion

Table 4.6 Case conclusion

Case	Input value	Best Algorithm
1	2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.) 5. Double extra large box size (54.80 cm. x 42.10 cm. x 33.50 cm.) 1. Small box size (31.12 cm. x 27.69 cm. x 3.81 cm.)	First-Fit Decreasing
2	5. Double extra large box size (54.80 cm. x 42.10 cm. x 33.50 cm.) 2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.) 4. Extra large box size (40.16 cm. x 32.86 cm. x 25.88 cm.)	First-Fit Decreasing
3	2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.) 1. Small box size (31.12 cm. x 27.69 cm. x 3.81 cm.) 5. Double extra large box size (54.80 cm. x 42.10 cm. x 33.50 cm.)	First-Fit Decreasing
4	1. Small box size (31.12 cm. x 27.69 cm. x 3.81 cm.) 2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.)	First-Fit Decreasing, and Last-Fit
5	2. Medium box size (33.66 cm. x 29.21 cm. x 6.03 cm.) 3. Large box size (45.40 cm. x 31.43 cm. x 7.62 cm.)	First-Fit

Chapter 5

Conclusion and Problem

5.1 Conclusion

This special problem focuses on using the bin-packing algorithm to solve the problem in finding the best loading items for the best benefit in using space and reducing the time in terms of loading items. The results of this application were displayed in 3-dimension.

In this application, there are three types of the containers and five models of box size for the users to select. The system will calculate the suitable place of the tray to fit the items in.

The benefits of program as the following are:

1) The application can simulate loading items before actual use.
2) The users can select the container types and the box sizes.
3) The users can input 20 set of items at a time.
4) The results of this application were displayed by using OpenGL.

From the experimental results in this program, it can be concluded that in most case using the First-Fit Decreasing algorithm can optimize the items arrangement and there is some spaces left in many user's requirement cases. But as for the First-Fit and Last-Fit algorithm, the solution depends on the ordering of user input, so the results were unpredictable.

5.2 Problems

1) Hard to communicate between GUI and OpenGL.

5.3 Limitation

1) Visual studio 6 can import only bitmap pictures.

2) The difference between OpenGL and GUI language.

3) Overlap might occur in the algorithms beside First Fit Decreasing.

5.4 Future work

1) Add more functions, more details of product, and more vehicle models. For example, the types of product and the delivery order.

2) Add the calculation of total weight in the application.

3) The results can show more detail of input parameters for example; quantity, delivery order, total weight, and the cost of delivery.

4) The users can store the information into the database.

5) Improve GUI.

6) Improve Last-Fit and First-Fit algorithm.

References

[1] Erick D. and Leon R. Kanavathy. Optimizing Three-Dimensional Bin Packing through Simulation. Proceedings of Sixth IASTED International Conference Modeling, Simulation, and Optimization. September 11-13 2006, pp. 60-85, ISSN 1790-0842.

[2] First Fit. Retrieved March 8, 2012, from http://encyclopedia2.thefreedictionary.com/First+Fit

[3] The Official Guide to Learning OpenGL, Version 1.1. Retrieved January 6, 2012, from http://glprogramming.com/red/chapter01.html

[4] GLUT - The OpenGL Utility Toolkit. Retrieved January 24, 2012, from http://www.opengl.org/resources/libraries/glut/

[5] Visual Studio 2008 Professional Edition System Requirements. Retrieved March 3, 2012, from http://www.ehow.com/list_7166006_visual-professional-edition-system-requirements.html

[6] Steps on how-to install and use the Visual Studio 2008. Retrieved February 25,2012, from http://www.tenouk.com/installusevisualstudio2008.html

Appendices

Appendix A

GLUT Installing Instruction

Install GLUT for Window 7

GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. [4]

1.) Download glut library and source from Internet and the zip file contains: glut32.dll, glut.h.

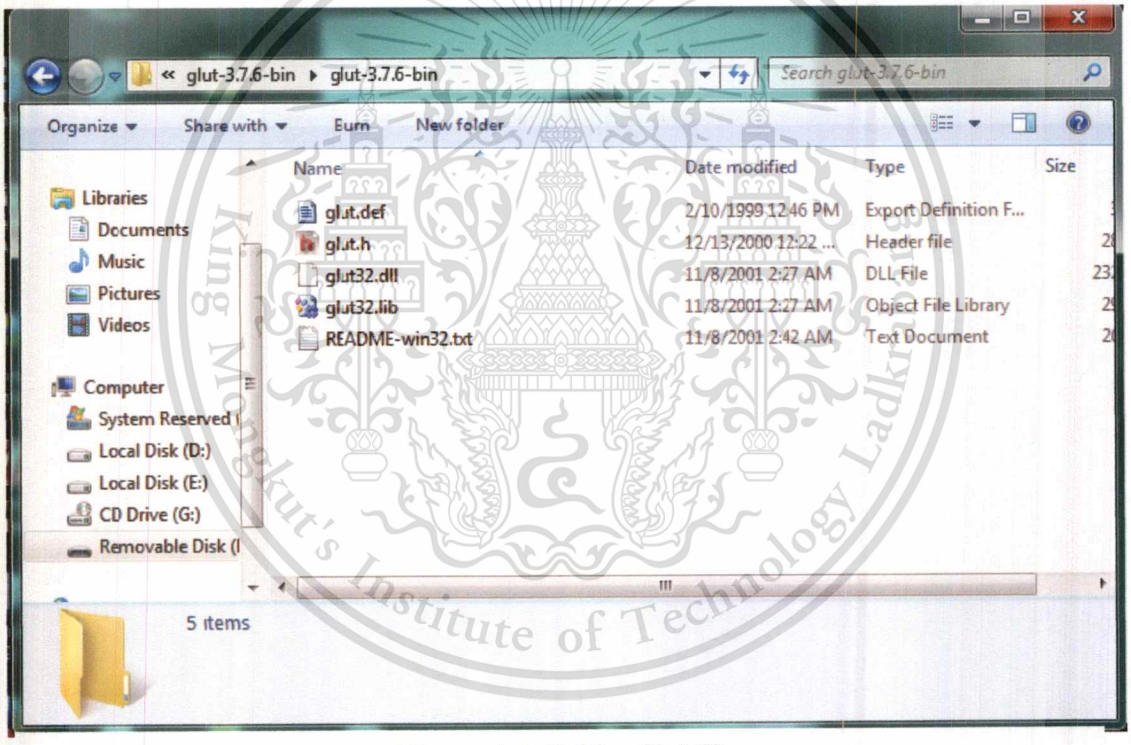


Figure A.1 Folder GLUT

2.) Move glut.h into C:\Program Files (x86)\Microsoft Visual Studio9.0\VC\include

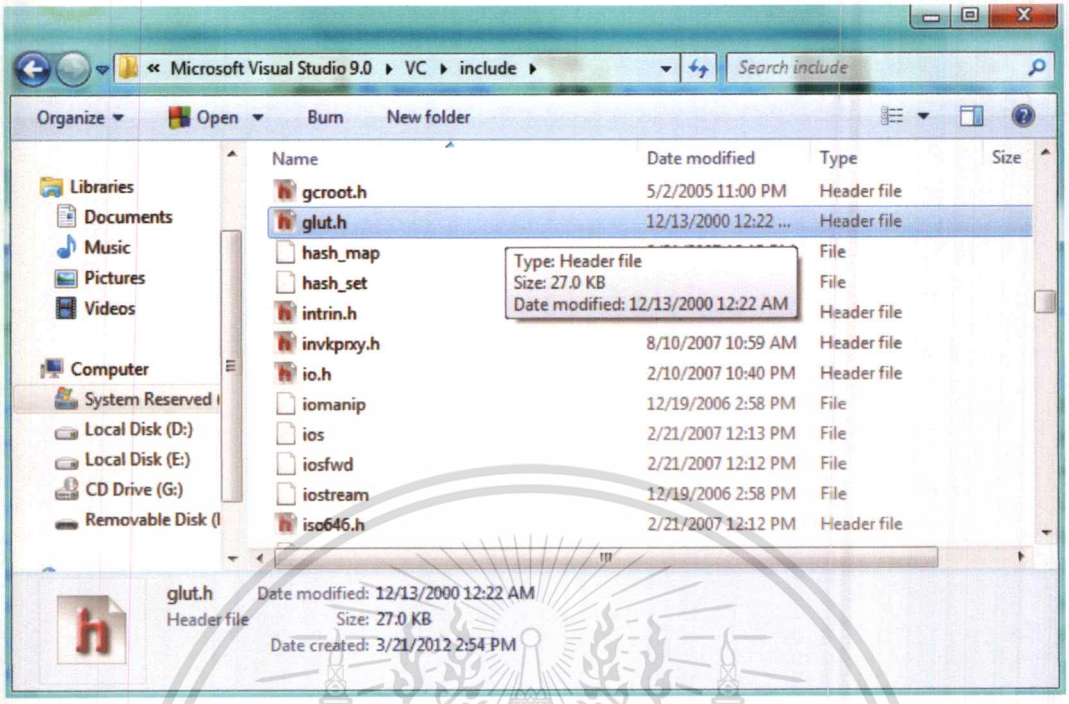


Figure A.2 Move glut.h

3.) Move glut32.lib into C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\lib\

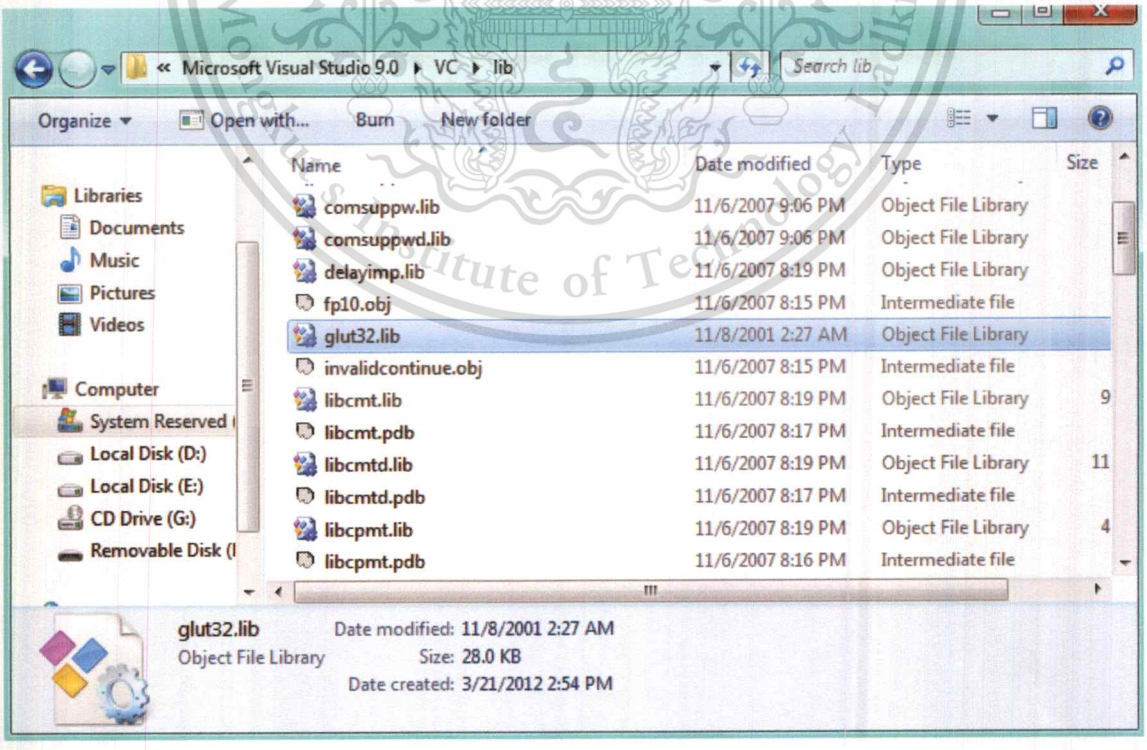


Figure A.3 Move glut32.lib

4.) Move glut32.dll into C:\Windows\System32\

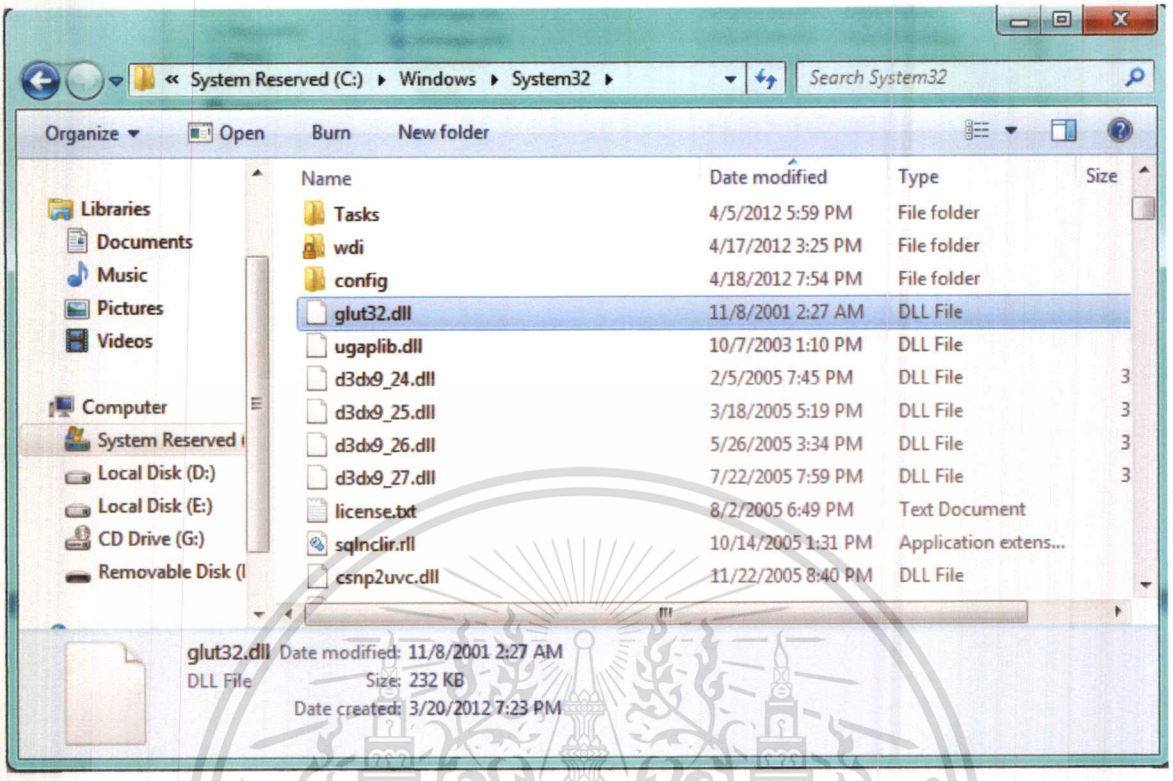
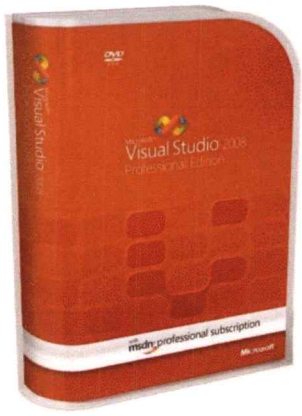


Figure A.4 Move glut32.dll

Appendix B

Visual Studio 2008 Installing Instruction



Visual Studio 2008 Professional Edition System Requirements [5]

Supported Architectures

- Visual Studio supports both 32-bit (x86) and 64-bit (x64) system architectures.

Supported Operating Systems

- The following operating systems are supported by Visual Studio 2008: Windows XP Service Pack 2 or above, Windows Server 2003 Service Pack 1 or above, Windows Server 2003 R2 or above, Windows Vista, Windows Server 2008.

Hardware Requirements

- The minimum requirements for Visual Studio 2008 Professional Edition are the absolute minimum required to run the application. They are a 1.6 GHz CPU, 384 MB RAM, a 1024x768 display and a 5400 RPM hard disk.

  The recommended requirements for Visual Studio 2008 Professional Edition represent the average computer that should be used to run the IDE. A computer with these hardware specifications should experience little to no performance issues with the most common features. They are a 2.2 GHz or higher CPU, 1024 MB or more RAM, a 1280 x1024 display and a 7200 RPM or higher hard disk

On Windows Vista, the hardware requirements are a 2.4 GHz CPU and 768 MB RAM.

Step of install Microsoft Visual Studio 2008 [6]

1.) Insert the DVD into the DVD drive, the auto run will be executed, displaying the following Windows form. Click the View ReadMe button to read the readme information. Then click the Install Visual Studio 2008 link to start the installation.

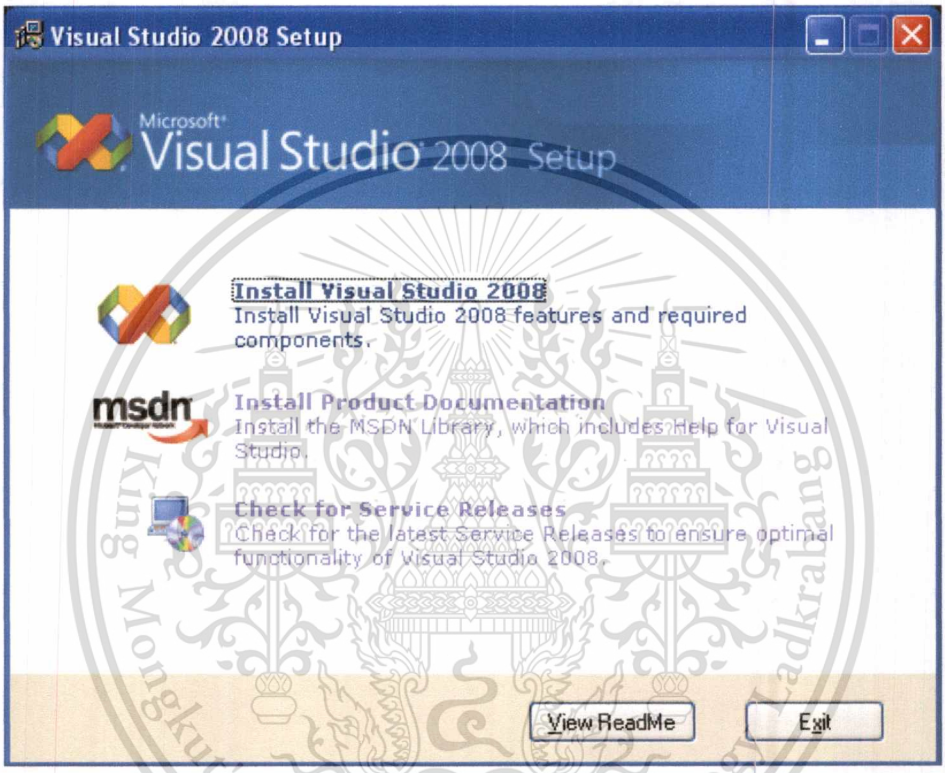


Figure B.1 Microsoft Visual Studio 2008 Professional Edition product

2.) The setup wizard will start copying needed files into a temporary folder. Just wait.

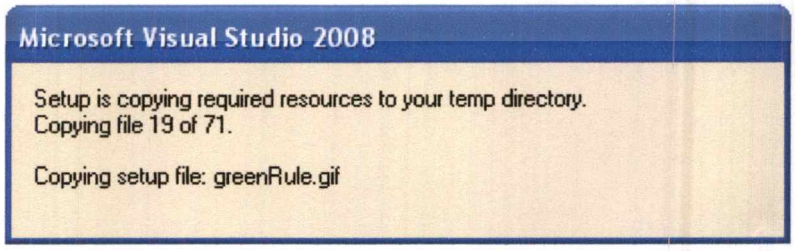


Figure B.2 Windows form to start the installation VS2008

3.) In the welcome setup wizard page you can enable the tick box to send your setup experience to Microsoft if you want. In this case we just leave it unchecked. Just wait for the wizard to load the installation components.

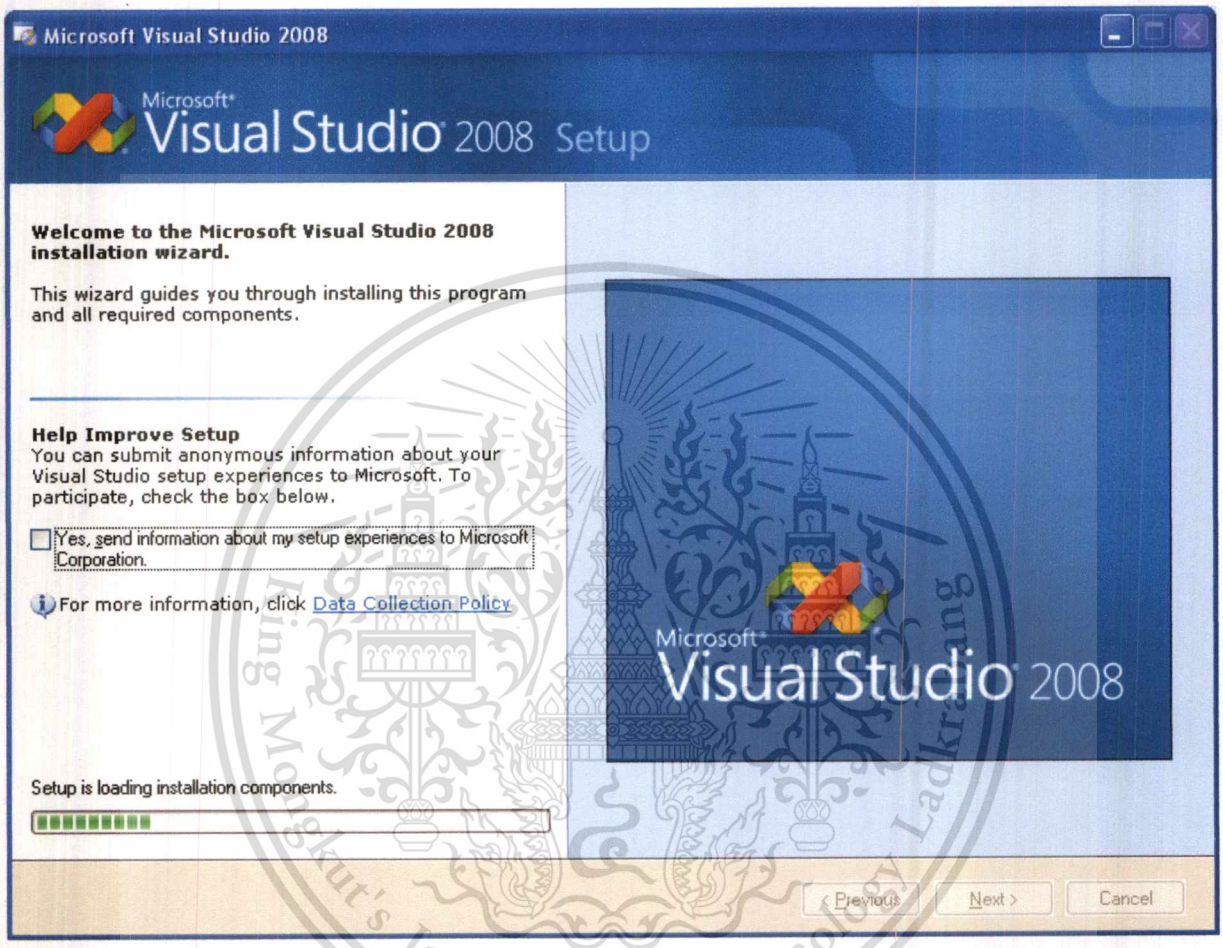


Figure B.3 Windows form to copy files into temporary folder

4.) Click the Next button to go to the next step.

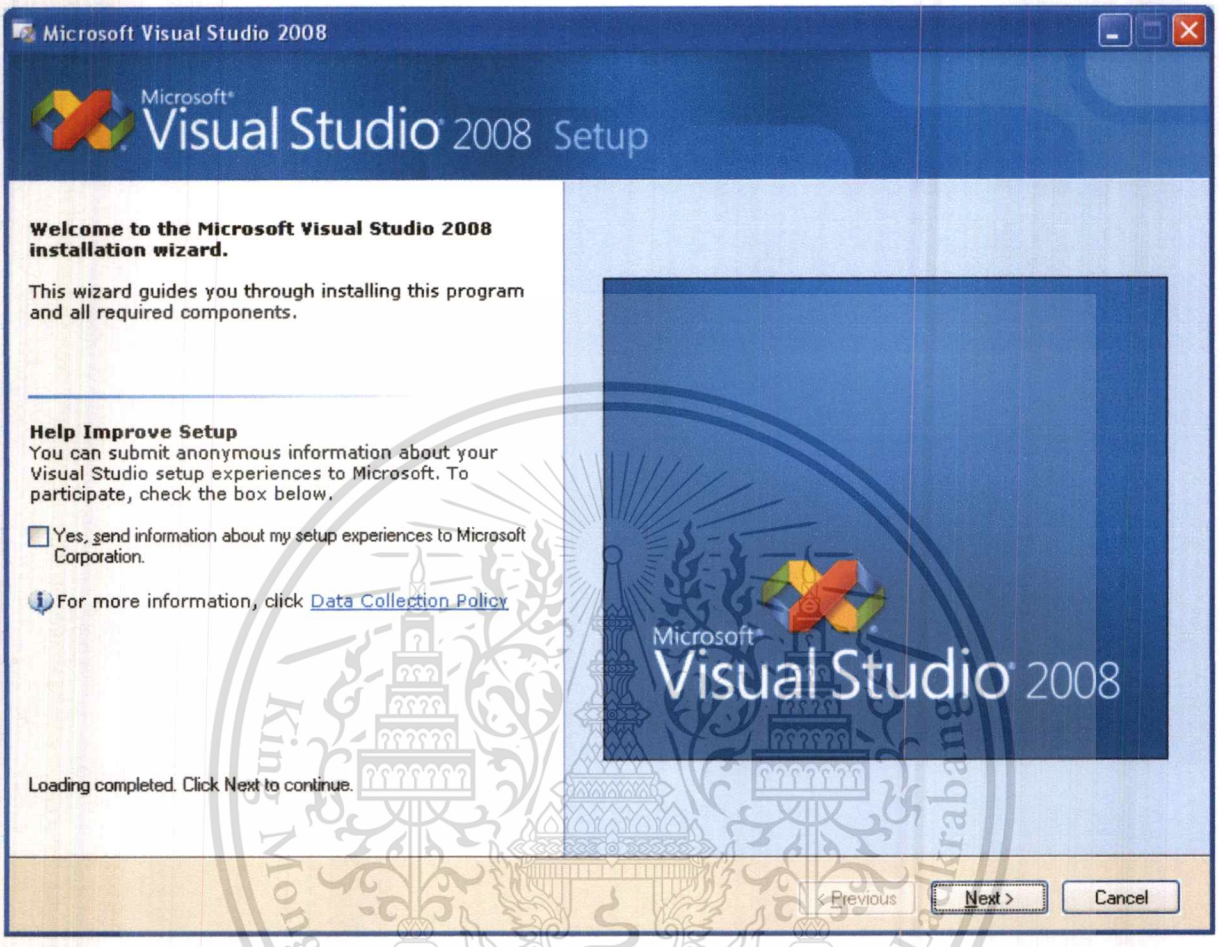


Figure B.4 The welcome setup wizard page

5.) The setup wizard will list down all the required components need to be installed. Any already installed components will also be mentioned. Notice that VS 2008 (version 8.x) needs .NET Framework version 3.5. Key in the Product key and accept the license terms. Then click the Next button.

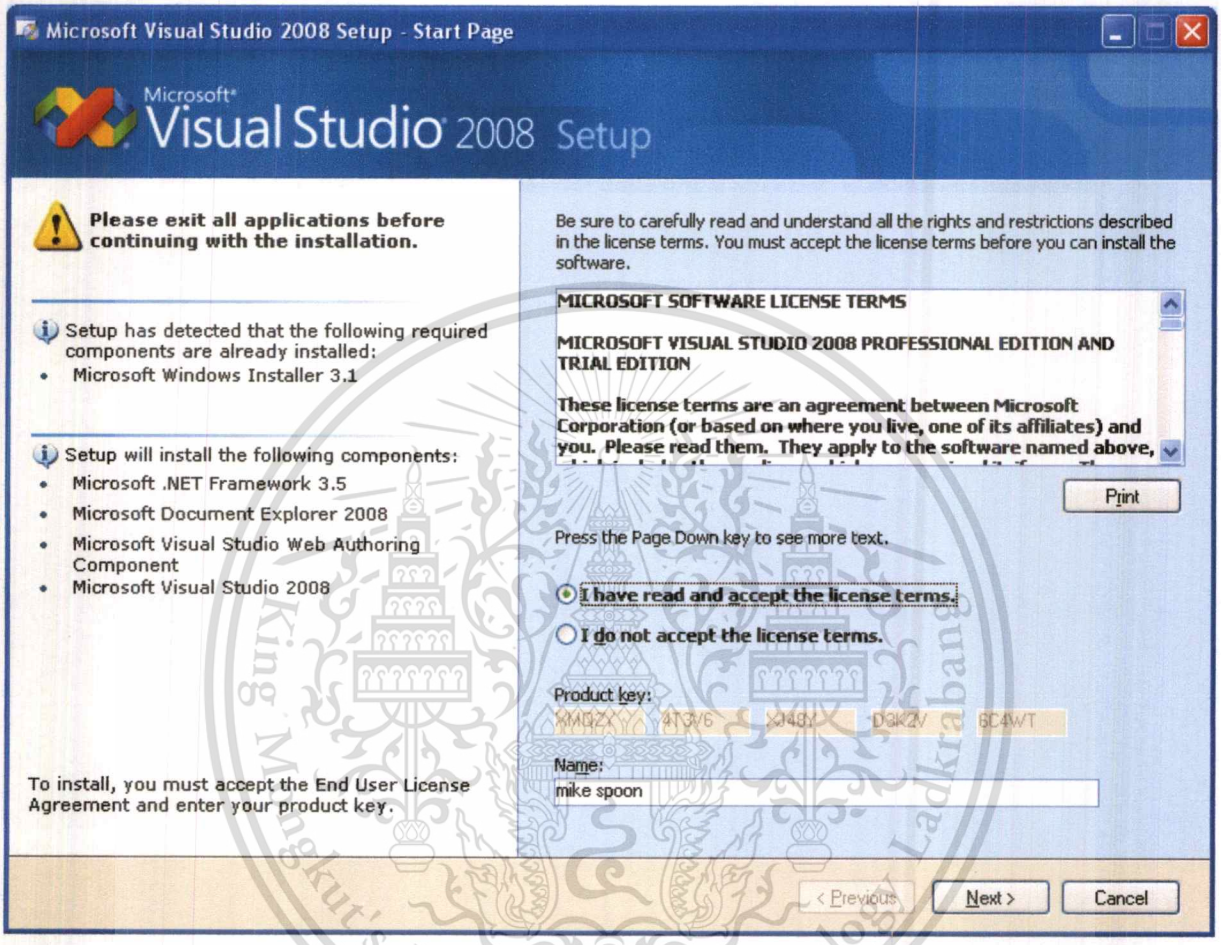


Figure B.5 The welcome setup wizard page in next step

6.) In the installation type, as usual we have three choices: **Default**, **Full** or **Custom**. In this case we select the Full installation type and accept the default installation path given. You can change the installation path and the required space for every installation type also displayed when we select it.

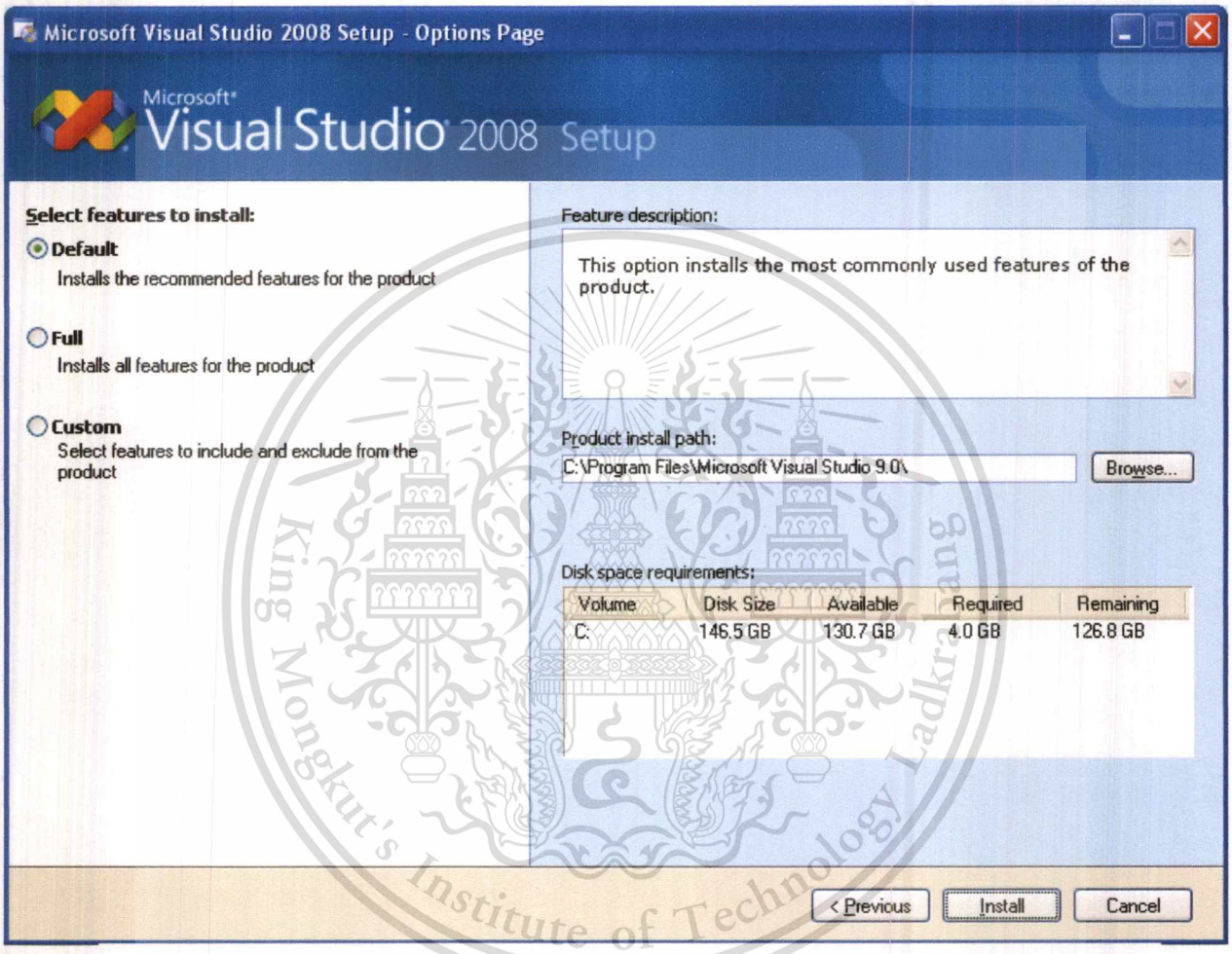


Figure B.6 The list of required components need to be installed

7.) In this case, select the Full and click the Install button. Full installation required around 4.3 GB of space.

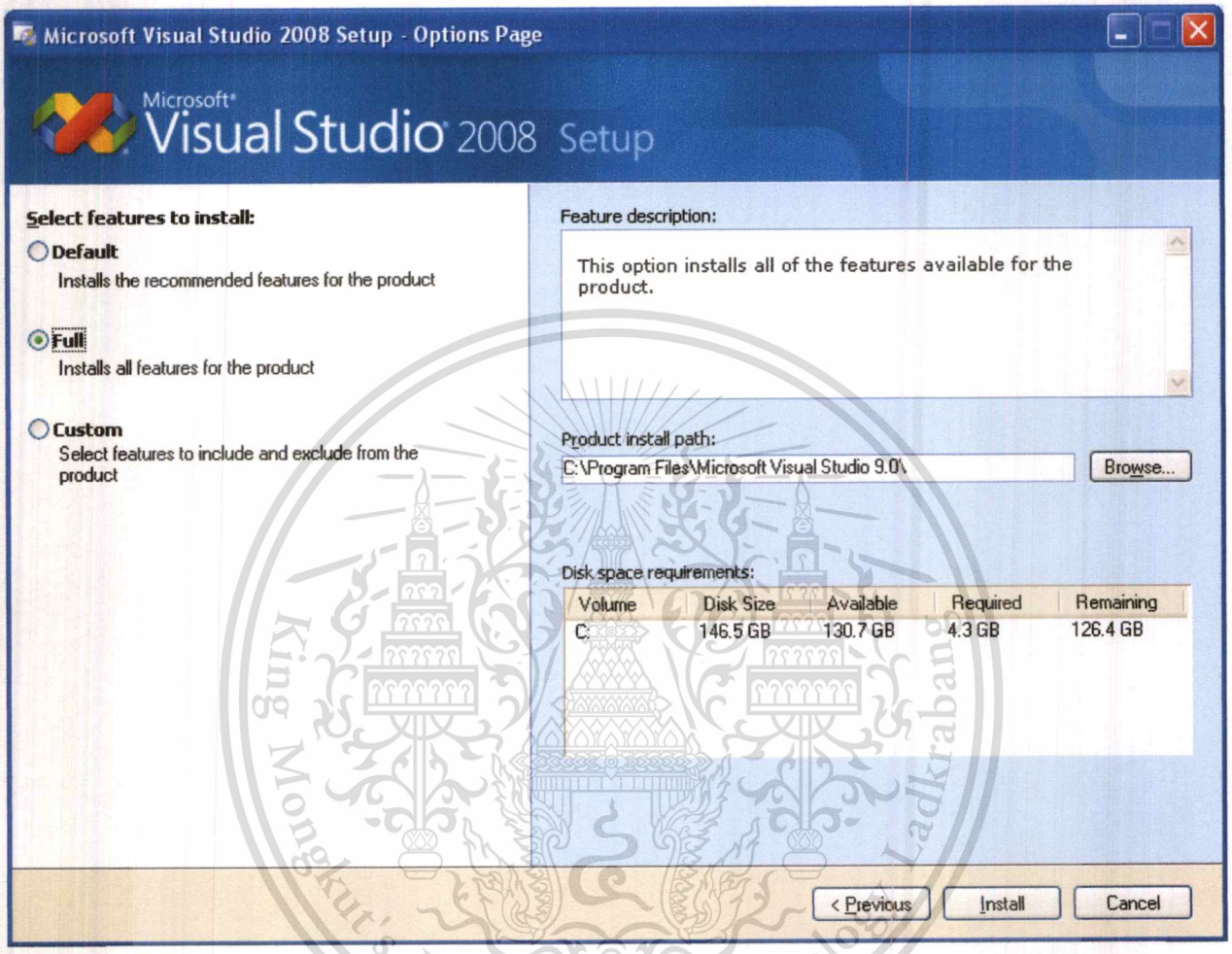


Figure B.7 The features type to install program

8.) The installation starts. Just wait and see the step-by-step, Visual Studio 2008 components being installed.

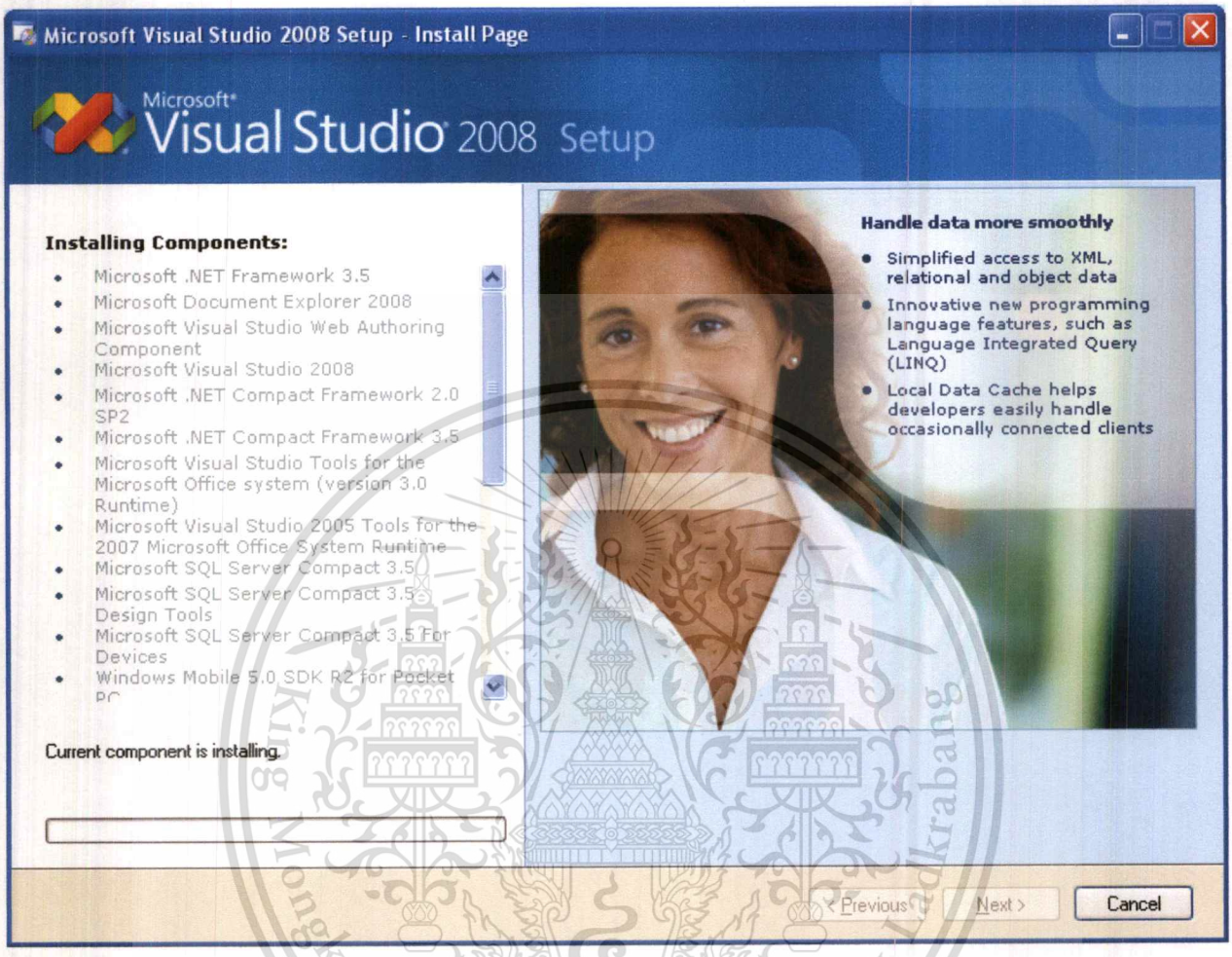


Figure B.8 Full type to installation

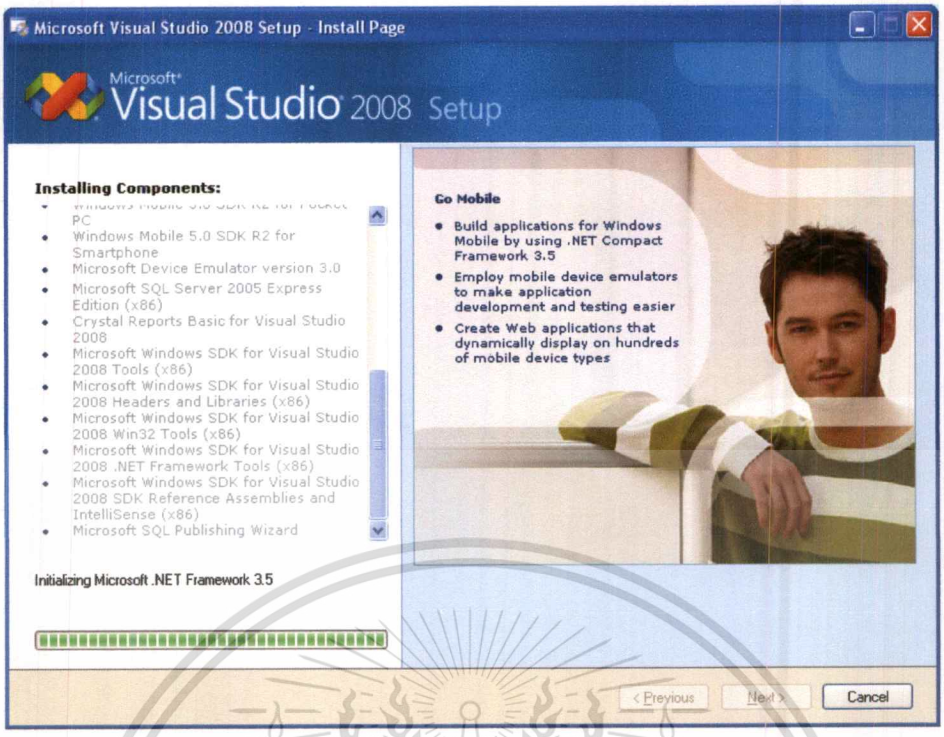


Figure B.9 Component to be installed

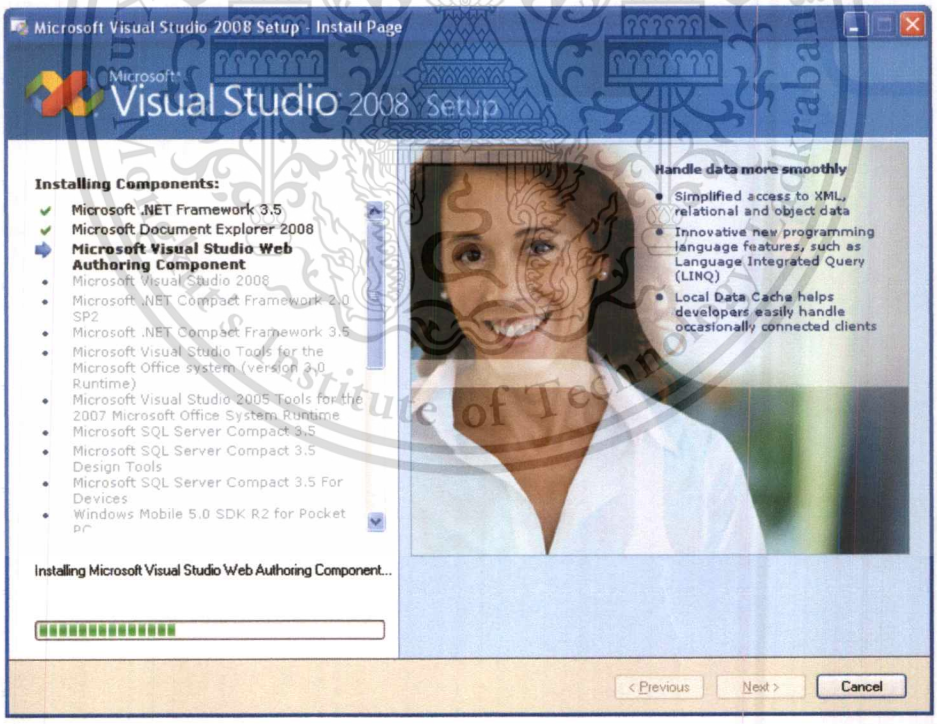


Figure B.10 Components to be installed (2)

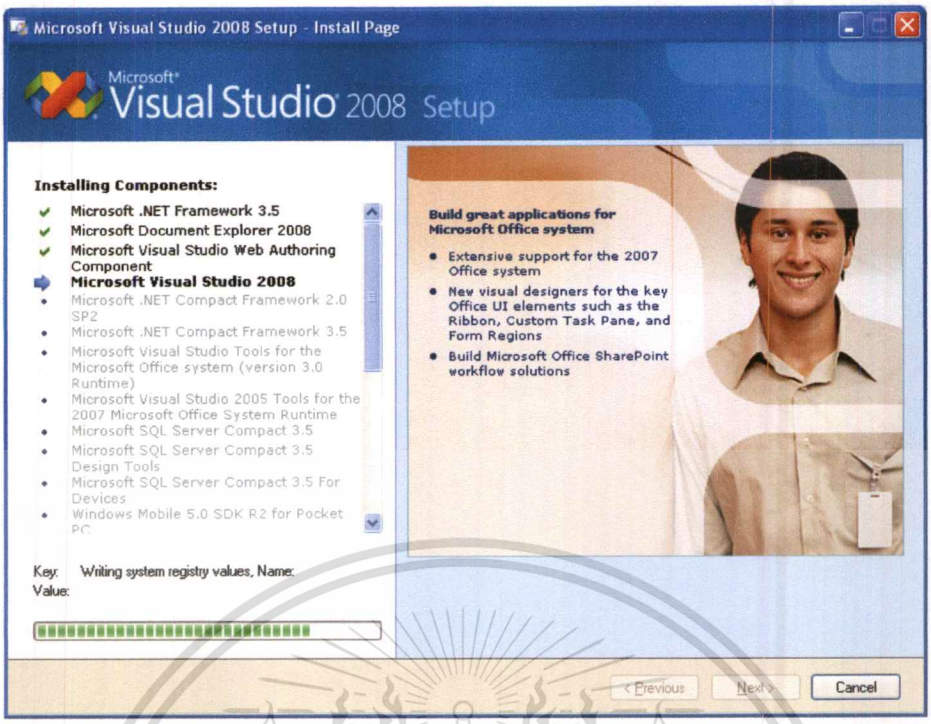


Figure B.11 Components to be installed (3)

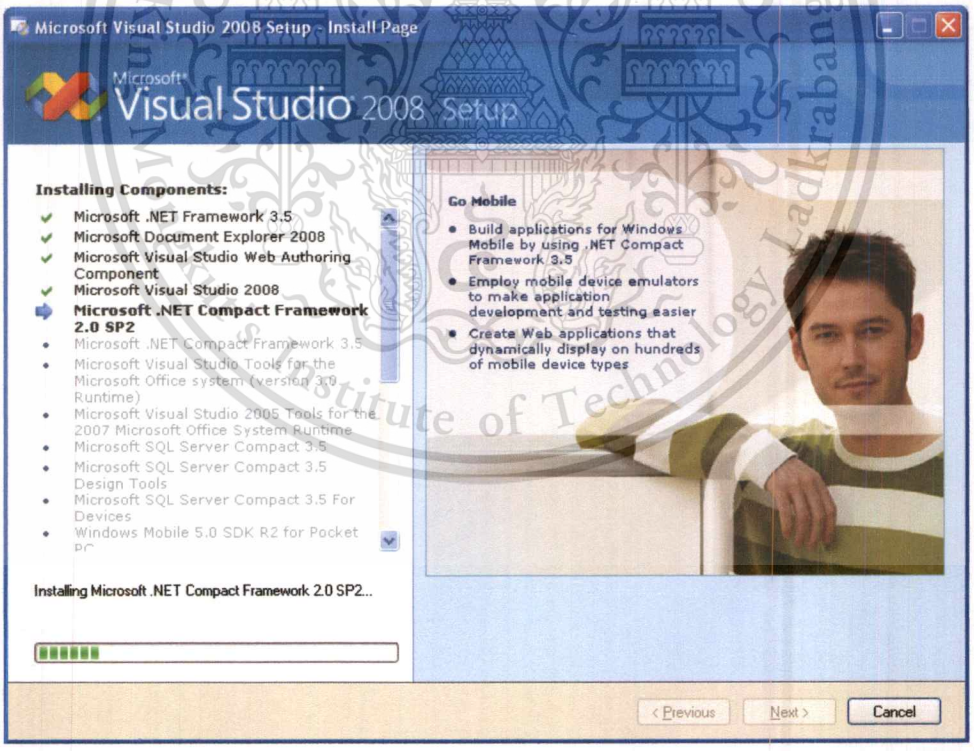


Figure B.12 Components to be installed (4)

9.) Any component that failed to be installed will be marked with the red cross mark instead of the green tick for the successful. After the installation is completed successfully, you can install the documentation (MSDN library) by following the instruction mention in the above Figure. In this case we just exit the setup wizard by clicking the Finish button.

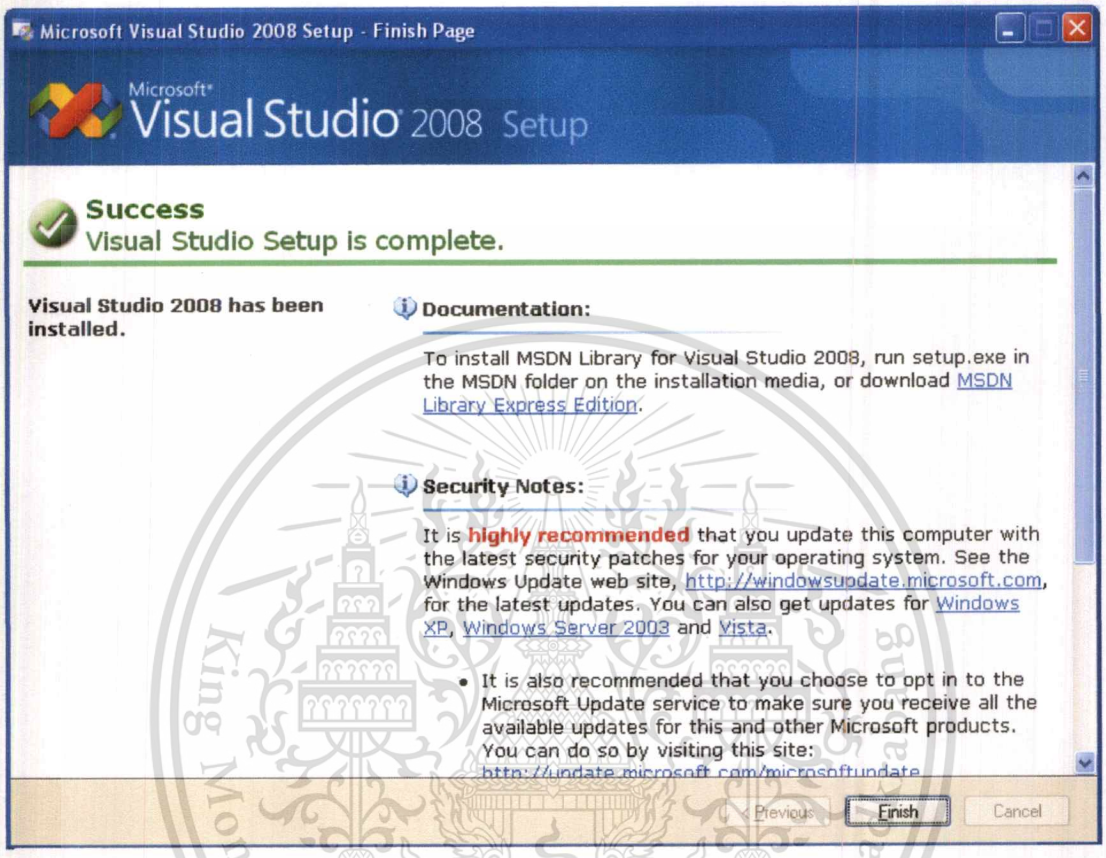


Figure B.13 Finished page to installed

10.) Click Restart Now to restart your computer.

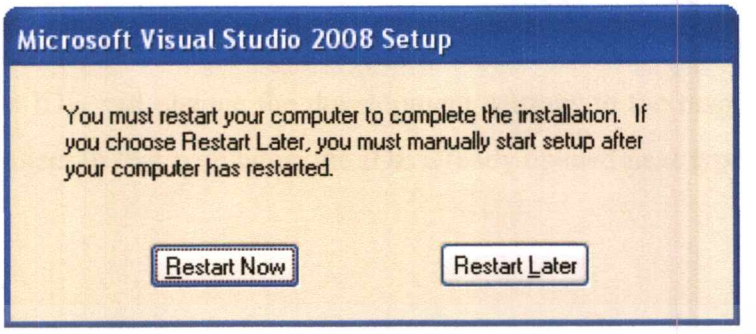


Figure B.14 Dialog to restart your computer

11.) The Windows Start menu for Visual Studio 2008 is shown below.

Figure B.15 The Windows Start menu for Visual Studio 2008

12.) Next we will try to build and run a very simple C++/CLI (C++ .NET) program. Launch Visual Studio 2008 by selecting Microsoft Visual Studio 2008 menu > Microsoft Visual Studio 2008 sub-menu. Depending on your programming needs, in this case we will select Visual C++ Development Settings. So, when VS 2008 launched, now and later, all the settings are default to VC++. The VS 2008 IDE will change the development settings to the respective type when we choose to create a project in that type when the IDE already opened next type. Select Start Visual Studio button.

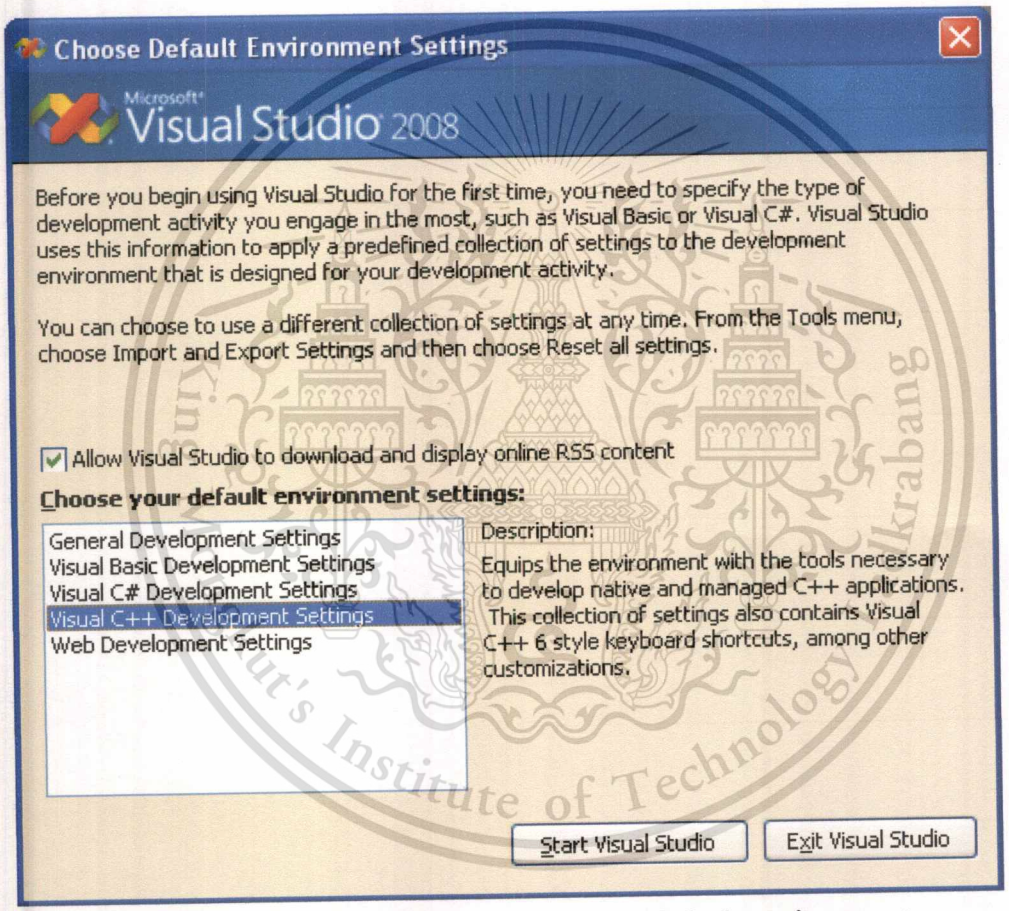


Figure B.16 Windows form to choose default environment

13.) The VS 2008 is configuring the development environments to the chosen one for the first time use.

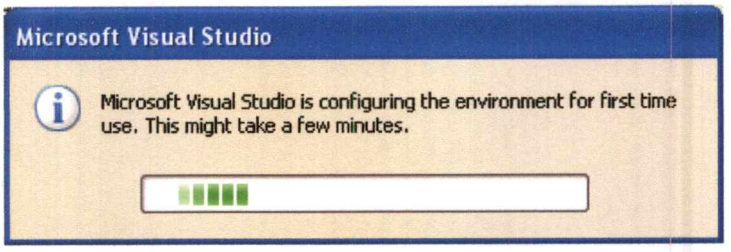


Figure B.17 Windows form to configuring the develop environments

14.) Well, the following is the VS 2008 IDE. Looks not so different to the VS 2005 other than the color.

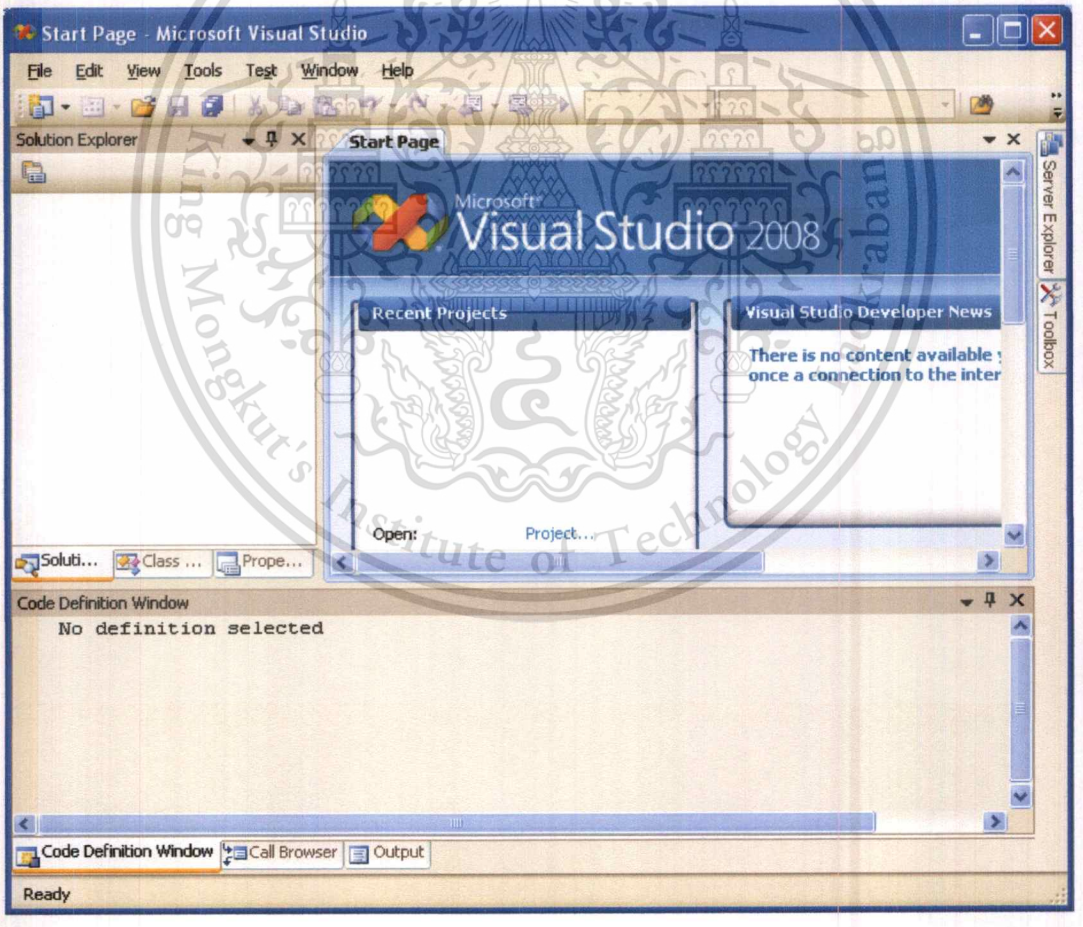


Figure B.18 Start page of Visual Studio 2008