FAKULTI SAINS KOMPUTER &TEKNOLOGI MAKLUMAT (PJJ)
UNIVERSITI MALAYA
SESI 2003/2004

PROJEK ILMIAH TAHAP AKHIR II
WXES 3182

VIRTUAL SOLAR SYSTEM

By
Chong Yee Wah
WQT 000092

Supervisor: Encik Mohd Nizam

TABLE OF CONTENTS

ABSTRACT

3D modelling is a three-stage process: First, obtain data and get it into the computer; second, use the computer to interact with it, just as manipulating a real object, but without annoyances like gravity and material strength; and third, output or store the modified data. Or, the data might subsequently be made 'real' again, by automatically fabricating an object using. Deriving a solar system is much challenging as it involves creation of nine different sizes of planets, obtaining the images that can be combined to form a realistic 3D meshes, importing the sound files that can pronounce the name of the planets and provide physically-valid movements for the planets.

In this Virtual Solar System's project, the goal is to create the interest for the younger generation especially in the primary school the existing of Virtual Reality world and to provide some basic information pertaining all the planets.

The methods used to create this animated 3D-model of the Solar System are as follows: -

- Data Acquisition – This includes obtaining colour-textured images of all the nine planets as well as recording and editing the pronunciation of each planet.
- Building the 3D model – This includes creation of various size for all the planets.
- Colour – Texture Mapping – This includes setting and properly aligning the 2D textured data onto the 3D model.
- Animation – This includes creating of revolving movement for the planets around the Sun.

CHAPTER 1

INTRODUCTION

CHAPTER 1 INTRODUCTION

1.0 PROJECT OVERVIEW

The Virtual Solar System is a Virtual Reality application. It was designed to allow user to navigate in the VR world. The Virtual Solar System consists of the Sun and all the objects that orbit around it. So far we know of nine planets in the Solar System which include Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune and Pluto.

The user not only can view the revolving movement of the planets around the Sun, they can also listen to the pronunciation of each planet and view the basic information of all the planets (example. name). This will create the interest of the amazing virtual reality world into the younger generation especially those in the primary school.

1.1 WHAT IS VRML?

This project is about to create a ***Virtual Reality Modelling Language*** (VRML) model to simulate the Solar System in 3D view. The procedure to create the physical models of a Solar System and a specimen is detailed in this report. So that, user can realised that all the objects are in 3 dimension.

Virtual Reality (VR) refers to technology used to provide computer-based experience that mimics real experience. The two primary characteristics of such experience are that it is three-dimensional and that we can interact with it. Sound is also desirable. The interactive 3D spaces created using VR technology are referred to as "worlds". The files that embody these spaces are also referred to as "worlds". In the case of VRML, they have the extension ".wrl". The virtual user that represents the user in the virtual world is called an "avatar".

VRML is an ASCII-based language used to describe VR worlds. Like HTML, it is an open, non-proprietary language that can be used by anyone without licensing. The International Standardization Organization (ISO) has officially adopted VRML. It supports text, graphics, animations, and audio. VRML browsers render VR worlds on the fly, an approach that permits the interactivity that VR demands. The behavior of the VR content may be triggered or modified by the user's interaction.

VRML provides the technology that integrates thee dimensions, two dimensions, text and multimedia into a coherent model. It functions most effective 3D file interchange format on Internet. This means that VRML serves as a simple, multi-platform language for publishing 3D Web pages.

1.1.2 The History of VRML

In this thesis, I am using the VRML2.0 standard, which provides an ideal solution to platform independent 3D modelling. However, before revealing the powerful features of VRML, it is important to know briefly why and how the VRML standard evolves and the VRML community which has done a great deal in the development of the standard, and in promoting VRML to make it a universal standard.

In 1994, Mark Pesce, was invited to present a paper at the First International Conference on the World Wide Web. Pesce and partner Tony Parisi had developed Labyrinth, a prototype three-dimensional interface to the Web. His presentation sparked a consensus: the conference attendees agreed there was a need for a common language to specify 3D scene descriptions. Work on the VRML specification began immediately following the WWW Conference. Consequently, an electronic mailing list was set up to assist in the discussion of the specification for VRML. Within a month, there were over a thousand members. The list membership quickly agreed upon a set of requirements for VRML: platform independence; extensibility; and the ability to work over low-bandwidth (14.4 kBps modem) connections.

After much deliberation, the proto-VRML community selected the Open Inventor ASCII File Format from Silicon Graphics, Inc. as the basis of VRML. The Inventor File Format supports complete descriptions of 3D scenes with geometry, lighting, materials, 3D user interface widgets, and viewers. It has all of the features that developers need to create highly interactive 3D applications, as well as an existing tools base with a wide installed presence. A subset of the Inventor File Format, with extensions to support networking, is the underpinning of VRML. Rikk Carey and Gavin Bell adapted the Inventor File Format for VRML, with extensive input from the www-vrml mailing list, and the support of Silicon Graphics Inc.

Some of the leading technical experts on the VRML mail list then formed the VRML Architecture Group (VAG). The VAG's goal was to foster consensus in the VRML community in order to develop a scalable, fully interactive standard for 3D shared worlds. The first meeting of the VAG was held August 1995 in Half Moon Bay. The second meeting was held October 1995, in Construct's offices in San Francisco. It was at this meeting that the VAG changed its strategic focus from technical design to process guidance. This shift resulted in a Request-For-Proposals (RFP) for VRML 2.0. The third meeting was held in February 1996 in Intervista's offices in San Francisco, to discuss the process relating to the VRML 2.0 RFP. During this meeting, the VAG proposed a formal process for the selection of a VRML 2.0 specification. The fourth meeting, in March 1996, was a return to Half Moon Bay. The VAG reviewed the RFP polling results and unanimously agreed that the Moving Worlds proposal would officially become the working document for VRML 2.0.

On August 4,1996, the official VRML 2.0 Specification was released at Siggraph 96 in New Orleans. At the fifth VAG meeting in New Orleans, the VAG finalised the formal announcement of the VRML 2.0 Specification, agreed to issue RFPs for a compress able binary format and an external authoring interface to the VRML 2.0 specification, and initiated the formation of the VRML Consortium.

1.1.3 The VRML World

A VRML world is made up of lots of simple shapes, such as cones and spheres, grouped together to form objects. The more shapes in the file, the more detailed the world, but at a cost of increasing the file size and the time taken by the browser to display the world. One reason why VRML in particular and virtual reality in general, has become more accessible recently is the increasing availability of relatively low cost, fast graphics cards and 3D accelerator chips, which allow complex worlds to be drawn quicker.

This VRML model (Virtual Solar System) is edited using Microsoft Notepad.exe. Then the .txt source file was converted into VRML format file by save it as .wrl file directly. To animate the environment in a Web browser, a plug-in is necessary: the VRML browser. Nowadays, VRML browsers are already distributed with some Web browsers, e.g. Netscape Navigator 4.x and Microsoft Internet Explorer 6.x. Alternatively these plug-in are accessible for free download from various Internet sites (e.g. Platinum: Cosmo Player Plugs In 2.1.1).

A VRML browser interprets any '.wrl'-files, which the Web browser has downloaded via HTTP protocol from the Web. A 3D scenery is created from this file, which is presented in a rectangular area of the browser window. The 2D viewpoint into the 3D world is predefined during initialization. However, the user can change any viewpoints with the help of VRML browsers' the control panel. This allows any possible navigation through the 3D VRML world. As VRML 2.0 supports animations and sensory, the user can interact with objects in the 'virtual environment'.

1.1.4 What is behind the VRML browser?

The browser uses 3D graphical libraries of the computer system such as OpenGL or Direct3D to present 3D visualization on screen. Beyond this software libraries hardware acceleration for 3D calculation are supported, if available.

The 3D scenery description in a '.wrl' file is set up in a hierarchical order. This hierarchy is called 'scene graph'; it is subdivided into groups and shapes. Shapes (and light sources and even sound) build a tree-like structure of objects, which have a parent-child-relation.

In other words, the VRML file is a list of nodes, which can be characterized as object nodes, grouping nodes and child nodes.

Without stressing syntactical aspects of VRML too much, the concept of grouping nodes should be explained here a little bit more precisely. On top of the 'Scene Graph' hierarchy we find grouping nodes, which manipulate subordinated child nodes (and other subordinated grouping nodes). At the very end of the branch there are elementary object nodes defining shapes like cylinders and spheres.

Any modification of higher-level nodes (e.g. rotation operations) reproduces themselves via child nodes towards the objects (principle of descent).

1.2 PROPOSED SCHEDULE FOR THE PROJECT

To facilitate that the whole project runs on time, a schedule is proposed to foresee the smooth running of the development of the whole system. The draft of the proposed schedule is drawn using a Gantt chart for a good look of the timeline proposed. Below is drawn a systematic timeline for building the Virtual Solar System.

MONTH/ ASSIGNMENT	AUG 2003	SEP 2003	OCT 2003	NOV 2003	DEC 2003	JAN 2004	FEB 2004
Literature Review	■	■	■	■			
System Analysis			■	■	■		
System Design			■	■	■	■	
Coding/Prototype				■	■	■	
Testing						■	■
Documentation					■	■	■

Chart 1: Proposed Gantt chart for the Project Schedule

A project schedule is planned to ensure the smooth running and implementation of this project. The project is planned such that it will involve constant developing so that the whole project will be developed following a planed schedule. The schedule is planned such that it coordinates all the aspects of the project so that it will be finished by the proposed time allotted.

CHAPTER 2

LITERATURE REVIEW

CHAPTER 2 LITERATURE REVIEW

2.0 INTRODUCTION

Today, in order to create an interesting and effective 3D modelling, it needs to have the following basic requirements:

- The model must be realistic. It must be able to accurately represent the physical shapes and appearances of the virtual.
- The model must be useful and it must be able to be exported to other application that demands its use. The model should be portable, in the formats that can be used by different applications, and scalable, for it to interact with possibly other scenarios and models in such applications.
- The model must be cost – effective. It is not an easy task to obtain a model that is both low in cost, in terms of time and resource. However, for a model to be effective used, it must be readily reproducible.
- The model must be simple and easy for the user to navigate in the virtual environment.

2.1 FEATURES AND CONCEPTS OF VRML

In this session, a brief overview of some powerful features will be discussed together with examples in helping to reinforce the concepts. Most of the features described here are used within the implementation of the project.

2.2 VRML FILE

A VRML file is a textual description of the VRML world. This files describes how to build shapes, the position of the shapes, their colour, etc. The extension of VRML file is .wrl. When a browser reads a VRML file, it builds the world described in that file, and as you move around, the browser draws the world. Normally, VRML file contains these items:

The VRML header

- Prototypes
- Shapes, interpolators, sensors and scripts
- Routes
- Nodes
- Fields and field values
- Defined node names
- User node names
- Comments

<u>The VRML Header</u>

The VRML header is needed in all VRML files. It provides information to browsers the type of the files. For example, consider the header below:

```
#VRML V2.0 utf8
```

This header means that the file is a VRML file, complaint with version 2.0 of the VRML specification, and the file is using the international UTF-8 character set.

Nodes

A VRML file contains nodes that describe the properties of the objects built. There are nodes describing shapes, colours, lights, viewpoints, position, orientation, etc. Usually, a node contains the type name of node, and a number of fields which describes the attributes of the node. For example, consider the example node below:

```
Cylinder {
Height 5.0
radius 2.0
}
```

As shown, this node describes a cylinder object with field's height and radius. Both fields have numerical values assigned which specify the values of the associated fields. Note that fields are optional within nodes since each field has a default value that is used by the browser if a value is not specified. For example, a default cylinder has a radius of 1 unit and height of 2 units.

Node Names

The nodes can be given names. The advantage is that when a node is given a name, it can be re-used in the world if necessary without having to define another object with the same attributes. This can save the processing time since only the original object is processed. This can also make updating easy throughout the world since if the original object is changed, the other instances which use the original will also be changed, instead of going through all objects and make modifications. For example, to define a name for the Cylinder node:

```
DEF my_cylinder Cylinder {. . .}
```

And in making an instance of my_cylinder, we can use the USE statement like this:

```
USE my_cylinder
```

Describing Shapes

A VRML shape has geometry, that defines its 3-D structure, and it has an appearance based on material, colour and texture. These attributes are represented by two fields, *geometry* and *appearance* in a Shape node. In VRML, several primitive shape geometries are predefined, such as box, cylinder, cone and sphere, which can be used straightaway. These basic shapes can be used to create more complex objects. However these objects have to be grouped in order to do so. Consider the following simple VRML example, which create a simple hut:

```
#VRML V2.0 utf8
Group {
children [
# Draw the hut walls
Shape {
appearance DEF Brown Appearance {
material Material {
diffuseColor 0.6 0.4 0.0
}
```

```
        }
        geometry Cylinder {
        height 2.0
        radius 2.0
        }
        },
        # Draw the hut roof
        Transform {
        translation 0.0 2.0 0.0
        children Shape {
        appearance USE Brown
        geometry Cone {
        height 2.0
        bottomRadius 2.5
        }
        }
        }
        ]
        }
```

In this example, one cylinder and a cone is used to build the hut. Moreover, this two shapes are grouped together using Group{} node. Note that the node that groups together the group's shape is called the *parent*. The shapes that make up the group are called its *children*. A group can have as many children within as possible. Moreover, groups can have other groups as children as well.

VRML Space

Since VRML deals with 3D building space, 3D coordinate system is required. The three axis, X,Y and Z are hence defined in VRML. In the 3D space, X is positive to the right, Y is positive upward, and Z is positive toward you in the space.

Events and Routes

VRML provides suitable resources for making the objects within a world dynamic. For example, a box within a world may move with a mouse click, trigger a sound and so on. In this case, wiring instructions are required. Wiring involves a pair of nodes to wire together, and also a wiring route, or path, between the two nodes. When a route is built between two nodes, one node can send information to the other nodes through the route. The information sent is called an *event*, which usually includes floating-point values, colour values, or 3D coordinate values. A more complex dynamic world can be built by routing multiple nodes together.

Node Inputs and Outputs

In order to send and receive events, some points within a node has to be able to connect to the outside. Most of the VRML nodes can be wired. For example, nodes that create lights have an input jack for turning them on and off. If this jack is wired, then the switching of the light can be controlled. Some nodes may have more than one input and output jacks as well, so that different attributes can be changed dynamically. A node's input jack is called an eventIn which sends events, while the output jack is called an eventOut, which simply receive events from the other end of the wiring.

Wiring Routes

A VRML circuit is built by describing a route from the eventOut of the sending node to the eventIn of the receiving node. When the first node activates the route by sending an event, the event travel to the second route through the route, and the node reacts. The type of reaction depends on several factors:

- The type of node receiving the event
- The node eventIn to which the route is wired
- The type and the values of the event
- The current activities of the node

For example, when an event TRUE is sent to the eventIn of a light node, and if the light is on already, no changes occur. However, if the light is previously turned off, it will be turned on by this event.

2.2.1 Building predefined Shapes

As mentioned in the previous session, shape has geometry and appearance, both defined by a Shape node. A shape's exact geometry and appearance are controlled by the choice of nodes and by the field values of these nodes. There are several basic objects predefined by VRML that can be used in Shape node. These objects include Box, Cone, Cylinder and Sphere nodes. Each of these nodes has one or more fields that can be modified to change the dimensions of the objects. The appearance of a shape is described by the Appearance and Material nodes.

The Shape Node

The Shape node has the following syntax:

```
Shape {
appearance NULL # exposedField SFNode
geometry NULL # exposedField SFNode
}
```

The value of the geometry can be changed by routing an event to the exposed field's implied set_geometry eventIn. When an event is received, the geometry field is set, and the new geometry can be sent out through the geometry_changed eventOut. The geometry field value can be an Appearance node and a Material node.

The Appearance Node

The Appearance node specifies appearance attributes and can be used a field value of the appearance field in the Shape node.

```
Appearance {
material NULL # exposedField SFNode
texture NULL # exposedField SFNode
textureTransform NULL # exposedField SFNode
```

The value of the material field can be a Material node. The default NULL value is equivalent to a default, glowing white material.

The Material Node

The Material Node specifies the material attributes and can be used as a value to the material field of the Appearance node. It basically contains fields used to modify the material of the object such as ambient intensity, diffuse and emissive color, shininess, and transparency etc.

The Primitive Shapes as geometry field value

There are some primitive shapes defined in VRML that can be acted as the value for the geometry field in the Shape node. Their nodes include:

- The Cone Node
- The Box Node
- The Cylinder Node
- The Sphere Node

The Group Node

The Group node is used to group specific object in the world to make them as one entity. There is a field children[] which specifies a list of child nodes to be included in the group. Usually, the children will be other group nodes.

2.2.2 Positioning, Rotating and Scaling Shapes

In VRML, shapes are built centered around the origin and hence any number of coordinate systems can be created with a world. Each coordinate system is positioned, or *translated* relative to another coordinate system. When a new coordinate system is relative to another, we say that the new coordinate system is a *child coordinate system* that is *nested* within the *parent coordinate system*. Moreover, that parent coordinate system can also be nested by other parent. Hence this parent-child relationship creates a family tree of coordinate systems. The root of this tree is the *world coordinate system*, which encompasses everything in the world. When a shape is translated, it is moved relative to the world coordinate system. The power of this concept is that, you can create each piece of the world independently. Each shape created is built within its own coordinate system. Then several such systems can be grouped together within a parent's coordinate system. The parent can then be grouped with other shapes to form a larger parent coordinate systems, and so on.

When a new coordinate system is created with a Transform node, it contains the appropriate fields for translating, rotating and even scaling the object in that coordinate system. For translation, we specify the translation distance in the X,Y and Z directions between the parent's origin and the new coordinate system's origin. When dealing if rotation, the values of the rotation field in the Transform node can be modified if required. However, we have to specify both the rotation axis and the rotation angle and direction in order to make an object to rotate in a coordinate system. The following table shows some common rotation axes:

Direction	Rotation Axis Values
To the right along the X axis	1.0 0.0 0.0
Up on the Y axis	0.0 1.0 0.0
Out along the Z axis	0.0 0.0 1.0

Table 1: Values of some rotation axes

The rotation angles are specified in radians. The angles can be positive or negative, depending on the direction you want to rotate the object with respect to the rotation axis specified. By default, the center of rotation is located at the origin of that coordinate system. However, it can also be specified.

Sometimes it may be necessary to scale the size of the object created. This can be easily accomplished in VRML by modifying the value of the scale field within the Transform node. VRML is able to scale a coordinate system, increasing or decreasing its size relative to their parent coordinate system. In this way, the shapes built within that coordinate system can also be scaled. To perform scaling, the scale factors of the three axis have to be specified. This means that you can scale an object differently on each axis. User can also specify a scalecenter other than the origin as it is the case with rotation.

2.2.3 Inlining Files

It is very common that different objects are created in different world and in different files. The most trivia advantage is that those objects can be reused easily. For example, it is a good idea to built a door in one VRML file, since there is a great chance that it will be used by several buildings, so that whenever a building needs that door, user can use that . Another advantage is that when one of the objects is complex and is made up of a lot of different objects; it will be hard to manage when user build all objects in the same file.

Any VRML file can be inclined using an inline node. The location of the file to be inlined is specified in the URL field in the Inline node. The URL tells the browser where to get the file, and can indicate files on the Web or on the local hard drive. Moreover the Inline node includes fields in which the size and center of a bounding box for the inlined files can be specified. The bounding box provides the benefit of delay reading, so that when the bounding box is outside a certain range from the viewer, the inlined file is not load until the viewer is within the range, so can save the time of downloading the files until it is really needed.

2.2.4 Using Faces in Building Shapes

The primitive geometry nodes predefined in VRML can be used to create various simple shapes, but they may not be enough if more complex objects are going to build. Sometimes it is quite difficult to construct irregular shapes using that primitive geometry. For example, in developing our project, we have encountered many cases that are not satisfactory when using basic shapes to construct because they are not realistic enough in this way. Fortunately, VRML provides a very flexible set of nodes for users to construct geometries using points, lines, and faces.

A VRML face is a flat shape like a square or triangle. A *face set* is a collection of these faces. In order to build a face, it is required to specify the perimeter of the face using a set of coordinates. The area within the face is filled and shaded to make it looks like solid. In building faces like this, we need to build a list of coordinate indexes each with a corresponding coordinate in X, Y, Z. The VRML browser will automatically connect those points together according to the order of the indexes. Then the perimeter is closed by connecting the last coordinate back to the first one. The area formed within will be filled. For example, if a square face is required, the coordinates for the four corners are shown in the following table:

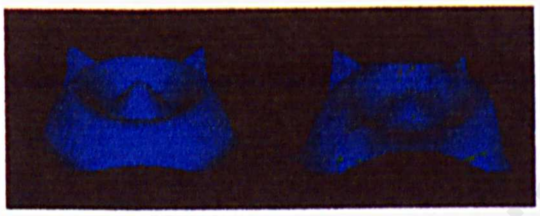
Coordinate Index	Coordinate (X,Y,Z)
0	-1.0 -1.0 0.0
1	1.0 -1.0 0.0
2	1.0 1.0 0.0
3	-1.0 1.0 0.0

Table 2: The coordinates for the four corners of a square

To build a square, a sequence of index can be defined such as, 0,1,2,3. The VRML browser will then follow the sequence to find out the corresponding coordinates, and then join them together in that order. It is recommended to build faces using coordinate indexes listed in counterclockwise order. Since only the front face is drawn to save processing time, there is a convention that the front of the face is the side of the face you see if its perimeter is traced in counterclockwise. Building 3D shapes using Indexedfaceset is simply building more faces and joining them together. VRML enables us to mark the end of one face and the beginning of the next by using the special coordinate index –1. With the help of Indexedfaceset, a lot of complex objects can be built efficiently and look realistic without having to join primitive geometry together.

2.2.5 Elevation Grids

In building virtual worlds, sometimes it is necessary to build a piece of land or a terrain. For example, in our project we have to build a landscape of the campus so that various buildings can be put onto it to constitute a realistic model. Although the IndexedFaceSet node provides an efficient way in building different shapes, the landscape may often covers a large area, and so making up an index with all corresponding XYZ coordinates will be a time-consuming and non-efficient process. However, VRML provides the ElevationGrid node designed specifically to build terrain grids and landscape. Using this node, we only have to specify the dimension of the terrain grid, the spacing between grid points and the elevation at each grid point. Faces are then automatically created for each grid square to create the terrain grid.



There are fields within the ElevationGrid node that are allowed to change the field values, such as the xDimension, xSpacing, zDimension, zSpacing and the height, which is the elevation. It is important to node that the spacing between grid points can greatly influence the quality of the shape created. For example, consider the following figures which shows a puddle splash built by using a 10x10(fig (a))elevation grid and a 40x40 elevation grid (fig(b)).

(a) (b)

Fig (a) A puddle splash having 10x10 grid elevation grid and (b) having a 40x40 grid

As shown in the above figures, it is quite obvious that 10x10 elevation grids is not enough to reveal the structure of the puddle splash. On the other hand in fig(b), a smoother, realistic model is the result of using more grids. However, one drawback in using more grids is that more faces are actually used to make up the model, and so more time is needed in processing, which may slow things down.

2.2.6 Level of Detail (LOD)

As more and more objects are built within the world, it will take the browser longer and longer time to display them on the screen. Consequently, if the model gets even bigger, the browser will spend almost all the time processing and have not enough to display the objects, and cause the world seems to be not moving at all. We must keep the balance between lots of details for maximum realism and quick drawing for maximum interactivity.

However, there is a technique which can solve the problem. In real life, when something is far enough, you cannot see as much detail as it is close to you. Therefore, in our virtual world, we can a high detail object with a low detail one when it is far away from the viewer. Objects that are really far away need not to be drawn at all. In this way, the browser will have less polygons to draw, and so can increase the interactivity. In VRML, this technique can be applied using the LOD (level of detail) node

In order to implement LOD on an object, first of all several different versions of the object with various details have to be created. After that, those shapes can be enclosed within the LOD node. There is a field within the LOD node called level, which are used to include those versions of the object. The highest detail version is listed first. There is also a level field within the node, which is used to provide values to the browser as to where to display the other versions of the object based on how far away the viewer is from the object. The distance between the viewer and a shape is called the range. The first value in the range field specify the distance at which to switch from the first to the second version of the shape, and so on.

2.2.7 Controlling Viewpoint

If a world contains only one object, it would be fine to let users to interact with the object freely without any help. However, when the world is getting bigger with more objects added to it, the user may find it hard to navigate in the world. The may 'lose their way', where they cannot find a particular object in the world. Therefore, a lot of time will be wasted in finding information within the world, and interactivity may lose most of its benefits. In order to overcome this, some sort of guidance should be available in order to help users in finding what they need quickly and efficiently. A good way is to set up predefined viewpoints in the world. Every time the world is loaded, the browser automatically positions the viewer at the predefined viewpoint position and the viewer is allowed to navigate from that point. The viewpoints can then act as reference points in the world so that the viewer still got some points to relate to if they are lost.

In VRML, viewpoints are controlled by Viewpoint nodes that specify a viewing location within a coordinate system. There is also a viewpoint stack, which is maintained by the

VRML browser. The Viewpoint node on top of the stack is the *current viewpoint* and is the one the browser uses to control the position and orientation of the viewer. Note that when the coordinate system of the current viewpoint changes, then the viewer within this viewpoint is also changed as well. There is a field called description within the Viewpoint node which can be used to give a name to a viewpoint. This name will be displayed by the browser in a viewpoint menu, which shows all viewpoints available. In our project, we have created viewpoints for all major buildings and sites so that when the viewer wants to look for a particular building, they can just choose it either in the menu button implemented ourselves, or use the existing one from the browser. Note that for some browsers, such as Cosmo Player, an option is available to set whether to fly through from the current viewpoint to the other or not. If so, the viewer will fly through the path between these two viewpoints. If not, the viewpoint will be changed immediately as a slideshow.

Apart from the Viewpoint node, there is also a node which is used the control the behaviour of the avatar. An avatar is a symbolic-virtual-world representation of a real-world person. The features of an avatar can be divided into two parts:

- features that describe what an avatar look like
- those that describe how an avatar can move around in the world

In VRML, the NavigationInfo node is available to control both aspects. There is one field that we have concentrated on in our project, which is the visibilityLimit field. This field controls a range which is visible from the viewer. Anything which is out of range will not be drawn. This technique can greatly reduce the number of polygons to be drawn at one time.

2.2.8 Colours and Textures

In order to make worlds more realistic, not only the shapes of the objects have to be looks similar, but also the appearance of those objects have to look like real objects too. As mentioned earlier, each Shape node contains a field called appearance, which basically controls how an object may look like through the colouring and brightness etc. VRML provides a node called Material node that specifies the material attributes and may be used as the value of the material field in an Appearance node. Some of the fields in a Material node include controlling for diffuse and emissive colour, transparency, ambient intensity and shiniess etc. Setting different values in these fields may result in quite different appearance. For example, user can make the object looks shiny, looks glowing, or even make it transparent.

Although in some cases, the Material node can give a fairly enough feeling of realism, there are cases which need extraordinary amount of visual details. Consider a tree. From far away, the tree looks like a large greenish blob. When we move closer, a trunk and leafy canopy can be seen. More closer and branches and clumps of leaves are visible and so on. In that case it is impractical to create a complex shapes in order to represent this extraordinary detail object. Therefore a technique called *texture mapping* evolves which allows us to take a picture from the real world and map it on any shape in VRML worlds. Texture mapping can save a lot of time. VRML also provides a lot of options in texture mapping just to make it easier to use.

VRML supports four of the most common image file formats for storing texture images: JPEG, GIF, PNG, and MPEG. Both colour images and grayscale images can also be used. Transparency images can be used as well. When an image with pixel transparencies is mapped to a shape, the pixel transparencies control transparency from point to point across the shape. Therefore, where on the image is transparent, the corresponding location on the shape is also transparent. The ImageTexture node specifies texture-mapping attributes and may be used as the value of the texture field in an Appearance node. One drawback of texture mapping is it depends on the architecture and hardware support of the machines.

2.2.9 Lighting

Lights in a VRML world are equivalent to the purpose as lights in the real world. Apart from the default headlight which is automatically created by the VRML browser, there are other types of light sources available and can be placed anywhere within the world. VRML supports three node types to control lighting effect: PointLight, DirectionalLight and SpotLight. Suitable use of these nodes can make the world more realistic and also can also act as devices to highlight some important spots of the world.

A *point light* is a light source that emanates light in a radial pattern is all directions, as if the light rays are coming from a single point. A point light is created by specifying its location in the world, its intensity and its colour. The light intensity can be varied from 0 to 1, and the colour of it is controlled by RGB colour. A *directional light* is one that seems to be situated in an infinity point in the world, so that the light rays are parallel and point to the same direction. The sun in real life is similar to the directional light in VRML. A directional light is created by specifying an aim direction for the light. The aim direction is defined the same way as defining rotation axes. The direction is the line between the origin and the second point which is under our control. Directional lights also support intensity and colour fields. Note that the default headlights from VRML browsers are actually a white directional light which aims forward and always illuminates anything in front of you.

A *spotlight* is a light aimed in a specific direction, and constrained so that its rays emanate within a light cone. In VRML, it is created by specifying the location, aim direction, and the cutoff angle and beam width to control the angular spread of the beam.

In our virtual world, besides the default headlight that we were using, we have also put into the world a spotlight which always follows the position of the viewer so that it's position and orientation is always the same as the viewer. This spotlight is acted as a torch in our world The cutoff angle and the illumination level of the light source can be changed using the EAI interface.

2.2.10 Adding background and Sound

To create efficient world backgrounds, VRML provides the Background node, which can be used to control the sky and ground colours and may be set up a panorama of background images as well. This node enables us to create an outdoors environment with using any polygons, and therefore can save drawing and processing time. Moreover, with the help of background binding, the background can also be changed according to some events or as time goes by as an animation using this method.

The realism of the world can also be increased by the integration of sound by using the Sound node provided by VRML. For example, a background music can be created. Sounds can also be added that are triggered by circuits, such as the sound when closing doors, etc. In order to add sound to the world, two components are involved, which are the *sound source* and the *sound emitter*.

The sound source provides a sound signal, and the sound emitter turns the source's signal into sound you can hear. In VRML, two types of sound format are supported, which are the wav file format and the General MIDI type 1 file. Moreover, there are field inside the Sound node which specify the *minimum-range ellipsoid* and the *maximum-range ellipsoid* that surrounds the Sound node's location. When the viewer is inside the minimum-range ellipsoid, the volume can be heard in full range. However, when the viewer is between the minimum and the maximum range, then the volume will be fading until it reaches the border of the maximum range ellipsoid. If the viewer is outside the maximum-range ellipsoid, then no sound can be heard at all.

2.2.11 Animation Position, Orientations and Scale and the use of Proximity

There are several components and nodes in VRML that allows us to put in animations into the worlds. These includes:

- The TimeSensor node – creates a clock that generates events to control animations
- The PositionInterpolator node - describes a series of key positions available for use in animation.
- The OrientationInterpolator node – describes a series of key rotations available for use in animation.

2.2.12 Sensors

The TouchSensor, PlaneSensor, SphereSensor and CylinderSensor nodes sense the viewer's pointing-device move, click, and drag actions for the shapes that are built within the same group as the sensor node. When these nodes are enabled, the viewer actions are converted into outputs that can be routed to other nodes to trigger animation or enable the viewer to manipulate shapes in the world.

CHAPTER 3

METHODOLOGY

CHAPTER 3 METHODOLOGY

3.0 DIFFICULTIES ENCOUNTER

The difficulties found while doing this thesis are:

- To create a vrml file, I need to think and imagine in 3D representation while having a 2D view. It took quite some time to understand the orientation of the axes system and that each object is being drawn to all directions from it's center of gravity.
- To search for the most appropriate colour texture image and information which represent each of the planets.
- To record and edit the sound files which pronounce the name of all the planets.
- Making a letter with elevation grid.
- Dealing with scale and distance ratio in the solar system, cause some are too small and too far.
- OrientationInterpolar, making a good rotation animation is hard when you have to use both xy or zx or zy axis.
- Sometimes, putting children inside children erroneously and trying to understand how the syntax works could be tedious.
- In lining a file at a height lower than or equal to that of the set avatar height cause the viewing level to drop to the bottom of the inline files.

3.1 TOOLS USED TO ENCOUNTER THE DIFFICULTIES

3.1.2 3DS Max 4.0

3D Studio Max, 3ds Max 4.0 is a professional object-oriented 3-D modelling, animation, and rendering program used for character animation, game development, and photo-realistic postproduction processing.

Users can create and save custom schemes that alter the basic configuration of the work environment. The modifier stacks let the user modify designs by undoing or altering any individual editing command in an object's history. You can reorder modifiers in the stack and apply stacks to other objects.

An important workgroup feature is the RPF file format, which contains not only rendered 3-D objects but also 3-D metadata, including camera information, Z depth, and other information. You can do extensive post processing on an animation without the need to rerender.

The main reason of choosing this software is that the user can create animation in 3D Max 4 environment and the files can be exported to VRML format.

3.1.3 Sound Forge 6.0

Sonic Foundry's Sound Forge 6.0 allows user to edit, record, encode, and master practically every form of digital audio, including WAV, AIFF, MP3, WMA, and even OGG files. Designed for professionals but intuitive enough for beginners, version 6 adds a bevy of new features, making it one of the most comprehensive and advanced two-track packages on the market.

Typical uses of Sound Forge include creating or fine-tuning loops and samples, recording audio from an external source (such as a microphone) or streaming media, and applying a desired effect on the entire track or just a piece of it. Sound Forge can also be used for mastering or remixing a recording and encoding into a compressed audio or video format.

The main reason of using this software is the user can record and edit the voice according to their satisfaction.

3.1.4 PhotoShop 7.0

Adobe Photoshop 7.0 is the world-standard image-editing, photo-retouching and Web-graphics solution application. With its integrated Web tool application, Adobe ImageReady, Photoshop delivers a comprehensive environment for professional designers and graphics producers to create sophisticated images for both print and the Web. Moreover, Photoshop 7.0 expands the definition of desktop image-editing by adding new support for vector-based drawing and editing, improved tools for producing Web graphics,and an enhanced user interface, all to your creative advantage.

Another two new retouching tools—the Healing Brush and Patch Tool—are best described as very smart cloning brushes that can make scratches, dust, and other imperfections disappear automatically. They work by blending the noise (or texture) of selected source pixels with the color and shading of the target area. The results are spectacular: The user can remove unwanted artifacts while preserving the color and shading details found in the original image.

The main reason of selecting this software as it allows user to edit and enhance the texture image which the user captured.

3.1.5 Cosmo Player 2.1.1

Cosmo Player is a high-performance, cross-platform VRML 2.0 client designed for fast and efficient viewing of virtual worlds. Navigate and manipulate 3D scenes and bring your Web experience to a new level.

Use VRML to fly through anatomy class, experience 3D data visualizations, or show off a CAD model. Cosmo Player is the premiere viewing client for VRML, with support for the latest standards.

Whether on the Internet or in an enterprise, Cosmo Player allows web content creators and applications developers to add visual and multimedia elements to their work.

3.2 PROJECT DEVELOPMENT STRATEGY

There are many types of development model in the software engineering such as Waterfall Model, Spiral Model and others. During the development of the Virtual Solar System, the prototyping model is selected since it allows all or parts of a system to be constructed rapidly to understand or clarify system needs.

3.2.1 Waterfall Model with Prototyping

This methodology is presented in Pressman [6] many other software engineering text books. As shown in the figure below, this model is chosen as it shows all the comprehensive steps on what happens during the development of the project and helps the developer to know the sequence of the event they should be expecting.

Some of the advantages of using this type of model with prototyping are:

- It ensures the developer is building the right system according to the specifications. This also enables the developer to conduct verification checks for the quality of the implementation.
- It is easy to associate and identify each milestone.
- This method is suitable to be used when there are uncertainties in the earlier stages of the project.
- Prototyping enables the early involvement of the developer and this makes the development more accurate.
- With is model; the development of the system can be back tracked if there is any errors found in the system. Any additions can also be edited without causing much time.

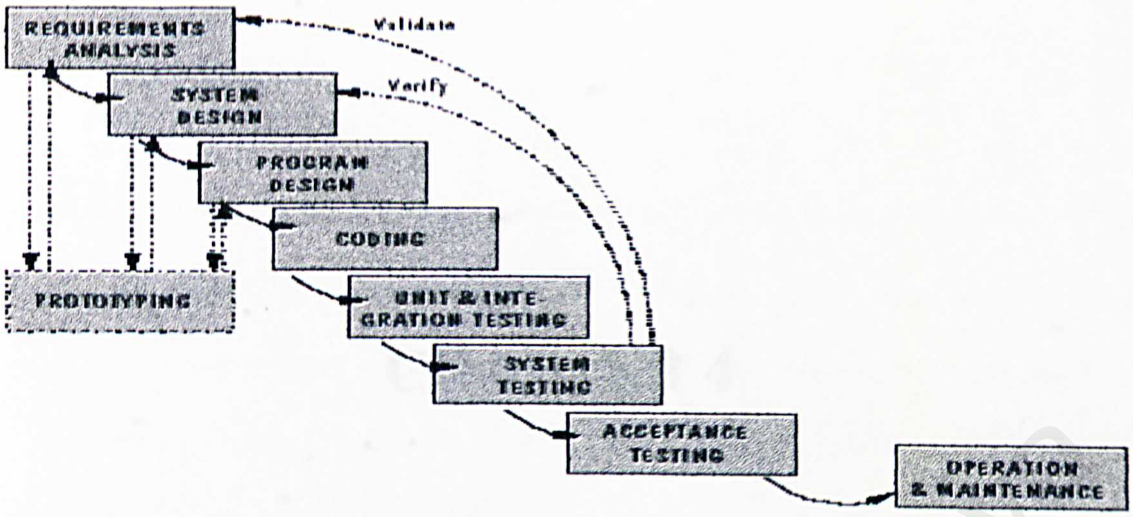


Figure 1: Waterfall Model with Prototyping

CHAPTER 4

SYSTEM ANALYSIS

CHAPTER 4 SYSTEM ANALYSIS

4.0 INTRODUCTION

System analysis phase involves information gathering and system's requirements identification. A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfil the system's purpose. Requirement describes a system's behaviour and activities. It expresses the system and object states and the transitions from one state to another.

Consequently, the primary goal of the system analysis phase is to distinguish the system's functional, non-functional requirements as well as the software and hardware requirements. The determination of these requirements are needed in order to support the stated functionality. Throughout the phase, the emphasis is to produce a user-oriented description of what exactly the system will perform.

4.1 FACT FINDING TECHNIQUES

There are various types of fact finding techniques. Here, interview and questionaires were chosen.

4.1.1 Interview

Interview is an important fact finding technique. An interview is a planned meeting to gather information from other person. It is a time consuming method but the best source of quality information.

4.1.2 Questionnaire

Questionnaires allow the collection of data in a large number of persons. A standard question format can give more reliable data. There are two types of questionnaires such as open-ended and close-ended questionnaire. Open-ended questionnaire allows interviewee write down their feelings, opinion on process. Closed-ended questionaire offers a specific response where interviewee only selects the answer by giving options. A questionaire was prepared for the VSS. (Appendix A).

4.2 SYSTEM REQUIREMENT

Requirement analysis is an important process. The acceptability of the system after it has been delivered depends on how well it meets the customer's needs and supports the work to be automated. If the analyst does not discover customers' real requirement, the delivered system is unlikely to meet the expectation.
In general, system requirements describe a system's behaviour. It can be divided into two modules that are functional and non-functional requirements.

4.2.1 Functional Requirement

A functional requirement describes the interaction between the system and its environment. The functional requirement also helps to describe how the system should behave if it has any stimulation. The important thing is that the questions addressed by the functional requirements must have the answers that are independent of an implementation of a solution to the problem.

4.2.2 Non-Functional Requirement

Non-functional requirements are set of constraints that a system must operate and the set of standards, which a delivered system must meet. This includes a graphical user interface (GUI), which provides a user-friendly interface. The system should be designed in such a way that users would not feel frustrated in using the system. Reliability is also a non-functional requirement, which the system should be reliable in performing its functions and operations.

4.3 SOFTWARE AND HARDWARE REQUIREMENT

Software Requirements: -

- 3D Max 4.0 (3D Modelling, animation and exporting to VRML 2.0)
- Adobe PhotoShop 7.0 (Texture)
- Sound Forge 6.0 (Sound)
- Netscape Communicator 4.7X with COSMO 2.1.1 browser (Viewing VRML model)

Hardware Requirements:

- Pentium III 800Mhz processor (recommended)
- At least 192 Megabyte RAM (recommended for using 3D Max 4.0)
- A 1.0 Gigabyte Hard Disk Storage Space
- Other standard computer peripherals

CHAPTER 5

SYSTEM DESIGN

CHAPTER 5 SYSTEM DESIGN

5.0 INTRODUCTION

After we analyse the system requirement, design should come into mind before we develop the system. Design is a creative process of transforming the problem into a solution or a description of the solution is also called design. It consists of two parts; the conceptual design and the technical design. A conceptual design tells customers exactly what the system will do, whereas the technical design allows the system builders to understand the actual hardware and software needed to solve the customer problem.

A good conceptual design should have the following characteristics:

- It describes the purpose of the system.
- It is independent of implementation.
- It is linked to the requirement documents.

On the other hand, the technical design usually includes the following:

- A description of major hardware components and their function.
- The hierarchy and function of the software components.
- The data structure and the data flow.

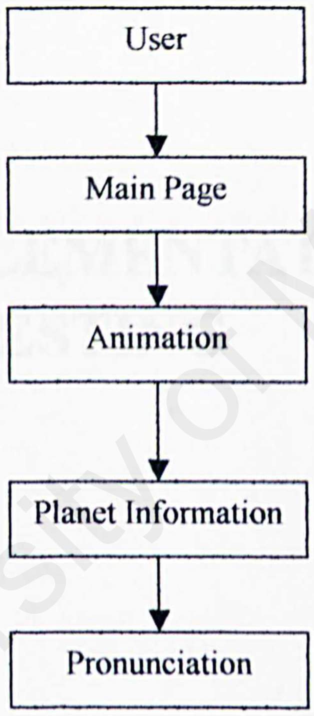
5.1 USER INTERFACE DESIGN

Graphical User Interface (GUI) is used for the user interface design as attractiveness is an important issue for this VRML application. In designing an interface, it must take into account some of the factors that influence the human nature. The Human Computer Interface (HCI) general principles of designing an interactive system is shown in Table

Principle	Description
Consistency	Consistent format for command input, input, data display, menu selection, and placing of the control objects.
Confirmation and Verification Message	Ask for verification for any non-trival destruction such as delete a record.
Recoverability	Ability of the user to take corrective action once an error has been recognized.
Forgive Mistake	The system should protect itself from user error that might cause it to fail.
Reverse Action	Allow user to return to the previous state (before change).
Functions Grouping	Categorize activities by functions and organize screen geography accordingly.
Simple Command Name	Use short and meaningful code to name commands. Short name is easy to memorize and reduce typing.
Responsiveness	How the user perceives the rate of command with the system. For example, the mouse pointer changes to hourglass or display a wait message when the system is processing data.
Context-Sensitive Help	Provide relevant help topic for current state when user needs the Help system.

5.2 DATA FLOW DIAGRAM

Data Flow Diagram graphically characterizes data process and flows in the system. It provides further understanding for the inter-relation between the systems and sub-systems. Generally, Data Flow Diagram describes in brief system's inputs processes and outputs. The diagram shows the planned context data flow diagram for "The Solar System".



CHAPTER 6

SYSTEM IMPLEMENTATION AND TESTING

CHAPTER 6 SYSTEM IMPLEMENTATION AND TESTING

6.0 INTRODUCTION

System implementation involves the translation of the software representation produced by the design phase into a computer-readable form. In other words, system implementation is a process that converts the system requirements and design into program codes. This phase at times involves some modifications to the previous designs into program codes. System testing verifies that a system solves the problem as defined by the requirements documents.

6.1 DEVELOPMENT ENVIRONMENT

The development environment has a very huge impact on the development of a system. The hardware and software chosen to develop the system are very import as they affect the speed of the development progress.

6.2 HARDWARE USED

The hardware used to develop this system is:

- Pentium 4 2.26GHz
- 256MB RAM
- 16X DVD-ROM Drive
- 24X CD Burner
- USB Flash Drive 128MB
- 40 GB Hard Disk
- Sound Blaster Compatible Sound Card
- Microphone
- USB Flash Drive 128MB

6.3 SOFTWARE USED

6.3.1 Tools for system development

The choice of development tools that are chosen plays a vital role in determining the usability of the system. Besides, development tools must be chosen with the development time in mind. It makes no sense to choose a very sophisticated tool but not managing to meet the milestones in time. With this in mind, the followings are the development tools used during the coding phase of "The Solar System":

Software	Module	Description
Windows XP	System Requirement	Operating System
Cosmo Player 2.1.1	System Requirement	VRML File Viewing
Microsoft Internet Explorer 6.0/ Netscape Communicator 4.7	System Requirement	VRML File Viewing
Adobe Photoshop 7.0	System Development	Image Editing
Sound Forge 6.0	System Development	Sound Editing
3D MAX 5.0	System Development	Animation Design
Microsoft Notepad	System Development	VRML Code Editing

6.4 PROGRAM CODING

Coding is a process that translates a detail design representation of software into a programming language realisation.

6.41 Coding principles

The following principles were applied during the implementation of "The Solar System":

- Coding conventions
  Coding conventions such as program labelling, naming conventions, comments and indentation should be adhered to.
- Readability
  Codes should be easy to understand. Adherence to coding conventions such as naming conventions and indentation contribute to program readability.
- Maintainability
  Codes should be easily revised or corrected. To facilitate maintenance, code should be readable, modular and as general as possible.
- Robustness
  The codes should be able to handle cases of user error by responding appropriately.

6.5 TESTING

Testing is a verification and validation process. Verification refers to the set of activities that ensure that the software correctly implements a specific function. On the other hand, validation refers to a different set of activities that ensuring the software has been built traçeable to user requirements. Software testing is a critical element of software quality assurance and represents the ultimate review of requirements specification, design and coding. The objective of testing can be stated as follows:

- Testing is a process of executing a program with the explicit intention of finding errors that are making the program fail.

- A good test case is one that has a high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.

Thus, testing is only successful when a fault is discovered or failure occurs as a result of testing procedures. Fault, identification is the process of determining what fault or faults caused the failure, and correction or removal is the process of making changes to the system so that the faults are removed.

6.51 System Testing

System testing is the major quality control measure during prototyping. System testing is to ensure that the entire system is validated, it must be combined with other system element such as hardware, end-user and databases. Testing is performed to ensure that the programs are executed correctly and confirms to the requirements specified. It provides a method to uncover logic error and for testing system reliability.

CHAPTER 7

SYSTEM EVALUATION

CHAPTER 7 SYSTEM EVALUATION

7.0 INTRODUCTION

During the development process, several problems were encountered in hardware, software, interfaces and logic errors in programming the required functions of the system. Some of them were solved at the end and some will be solved in due time. The system strengths and limitations were evaluated by variety of users. Improving of the present system and potential enhancement are based on suggestions and evaluations result form these users.

7.1 PROBLEM ENCOUNTER AND ALTERNATIVE SOLUTIONS

During the process of developing the Virtual Solar System, various problems were encountered. Below are the problems faced and how they were handled:

7.11 No experience in VRML programming.

Many errors occurred during the programming process and need a lot of time to debug. Usually refer to reference books, surf internet and get assistance from friends.

7.12 Lack of knowledge in software applications.

Generally refer to reference books, online help and friends to overcome the difficulties especially during the animation process.

7.2 SYSTEM STRENGHT

7.2.1 User-friendliness

The Virtual Solar System interfaces are developed through VRML which will work well with the Internet Explorer browser and Cosmo Player. Thus, the Graphical User Interface (GUI) provide users with an easy environment to work on. A standard set of GUI control objects such as command buttons has been applied. Furthermore, the user will only need to use the mouse to click on the object to view the results without keying any command or value.

7.2.2 No installation required

The user can run the program directly from the source without installing the whole program. It will save user's access time to the program and also save the storage space. The total size of the program is approximately 1.08MB.

7.2 SYSTEM LIMITATION

Every System has its limitation. Some of the limitation will be highlighted as follows:

- The system is developed only for educational purpose.
- It is not a web-based application.
- The information provided is limited.
- Basic but dull in interfacing.
- No commercial value.

7.3 FUTURE ENHANCEMENT

There are a lot of new features that can be added into the system to make it more attractive and advance. Below is the list of some features that can be advanced:

- Expand the system towards commercial orientated.
- Enhance the system to be available online.
- Widen the information scope of the Virtual Solar System.
- Provide online help.
- Allow advertisement to be published in the web site.

CHAPTER 8

CONCLUSION

CHAPTER 8 CONCLUSION

In conclusion, it can be said that the Virtual Solar System has met its objective as a Virtual Reality application for educational purpose. The system is easy to learn and use, yet attractive to capture the attention of younger generation.

Development the system is not an easy task. But, with the proper selection of software used, the development has been made much simpler. 3ds Max was used to design the animation as it would be much easier and time saving compare to the usage of VRML. Furthermore, the 3ds Max file can be converted into VRML file and this will definitely save a lot of storage space.

However, due to the time constraint, only minimum requirement have been included. Thus, future enhancement has been proposed to improve the system. In addition, much experience has been gained, new knowledge acquired and opportunities available to practice some project management and communication skills.

In view of the above, the objective for a final year project has been achieved; which is to give undergraduates an opportunity to undergo different challenges while allowing them the opportunity to experience the whole system development process.

APPENDIX A

QUESTIONAIRE

APPENDIX A

QUESTIONNAIRE/ BORANG SOAL SELIDIK.

(Primary/Rendah)

Date/*Tarikh*:________________

This form has 2 pages /*Borang ini mempunyai 2 halaman.*

Please answer all questions accordingly and **tick** only **one** answer/
Sila jawab semua soalan dengan baik dan ***tandakan*** *hanya* ***satu*** *jawapan sahaja.*

1) Have you used a computer?/*Adakah anda pernah menggunakan komputer?*

☐ Yes/Ya ☐ No/Tidak

2) If answer for (1) is Yes,where do you usually use a computer?/
Jika jawapan bagi (1) adalah Ya,di manakah anda selalu guna komputer?

☐ House/Rumah

☐ Friend's house/Rumah kawan

☐ School/Sekolah ☐ Cyber cafe/Kafe siber

3) What kind of operating system does your computer use ?/
Sistem pengendalian jenis manakah yang komputer anda gunakan?

☐ Windows ☐ Unix

☐ Macintosh ☐ Linux

4) Why do you use a computer?/*Kenapa anda gunakan komputer?*

☐ Studies/Belajar ☐ Games/Permainan

☐ Chat,e-mail/Chat,e-mel ☐ Surfing/Melayari Internet

5) How well do you understand the term 'multimedia' in terms of education?/
Apakah kefahaman anda mengenai 'multimedia' dari segi pembelajaran?

☐ Very well/Sangat baik ☐ Quite well/Baik

☐ Moderate/Sederhana ☐ Unsure/Kurang pasti

6) Have you heard of multimdia teaching packages?/
Pernah dengar tentang pakej pembelajaran multimedia?

☐ Yes/Ya ☐ No/Tidak

If yes, please specify/
Kalau ya,sila terangkan

____

7) Have you tried using it?/*Pernah cuba gunakan pakej pembelajaran?*

☐ Yes/Ya ☐ No/Tidak

If yes, why?*Kalau ya,kenapa?*_______________________________

8) Given the chance,would you use this package?/
Adakah anda akan gunakan pakej sebegini jika diberi peluang?

☐ Yes/Ya ☐ No/Tidak

9) Given the chance,would you use this as your project?/
Adakah anda akan gunakan pakej sebegini dalam projek jika diberi peluang?

☐ Yes/Ya ☐ No/Tidak

10) Is it seay to understand this project solution?/
Adakah senang menggunkan solusi projek ini?

☐ Yes/Ya ☐ No/Tidak

APPENDIX B

SAMPLE SOURCE CODE

APPENDIX B

SOURCE CODE FOR THE VSS WELCOME PAGE

```
#VRML V2.0 utf8

Viewpoint { position 0 0 20 }

DEF B_Black Background {}

Transform {
 rotation 0 0 1 1.5708
 children [
  DEF T_Back Transform {
   children [
         DEF B_Ani Background {
          skyAngle [ 0.2618, 0.5236, 0.7854, 1.0472, 1.3090, 1.5708, 1.8326, 2.0944,
2.3562, 2.6180, 2.8798, 3.1416 ]
          skyColor [ 1 1 1, 1 1 1, 1 0.75 0.75, 1 0 0, 1 0.5 0.5, 1 1 0, 0 1 0, 0 1 1, 0 0 1, 1 0
1, 1 0.75 1, 1 1 1 ]
         }
   ]
  }
 ]
}

DEF TS_Time TimeSensor {
 cycleInterval 5
 enabled FALSE
 loop TRUE
}

DEF OI_Back OrientationInterpolator {
 key [ 0 0.25 0.5 0.75 1 ]
 keyValue [ 1 0 0 0 , 1 0 0 1.5708 , 1 0 0 3.1416 , 1 0 0 4.7124 , 1 0 0 0 ]
}

Anchor {
 url "orbitingplanet.wrl"
 description "Welcome To The Solar System"

 children [
         Shape {
          appearance Appearance {
                material Material { transparency 1 }
            }
            geometry Box { size 50 20 0.001 }
            }

DEF TS_Ani TouchSensor {}

DEF SoundWelcome Sound {
        minBack 5
        minFront 5
        maxBack 50
        maxFront 50
      source DEF SoundWelcome AudioClip {
                url "welcome.wav"
                }
              }


 ]
}

#DEF TouchUranus TouchSensor { },


Transform {
 translation 5.0 -1.0 0
 children [
      Shape {
       appearance Appearance {
           material Material { diffuseColor 1.0 0 1.0 }
       }
       geometry  Text {
                 string  "To The"
                 fontStyle DEF FS_Text FontStyle {
                  size 2
                  justify "MIDDLE"
                  style "BOLDITALIC"
                  family "PERPETUA"
                  }
          }
      }
 ]
}

Transform {
 translation -5.0 0.0 12
```

```
children [
     Shape {
      appearance Appearance {
           material DEF M_W Material { diffuseColor 0.7 0 0 }
      }
      geometry Text {
                string  "W"
                fontStyle FontStyle {
                size 3
                justify "MIDDLE"
                style "BOLDITALIC"
                family "VERDANA"
                }
      }
        }
]
}

Transform {
 translation -3.8 0.2 10
 children [
      Shape {
       appearance Appearance {
            material DEF M_E1 Material { diffuseColor 0 0.7 0 }
       }
       geometry Text {
                string  "E"
                fontStyle FontStyle {
                size 3
                justify "MIDDLE"
                style "BOLDITALIC"
                family "VERDANA"
                }
         }
      }
 ]
}

Transform {
 translation -3.0 0.4 8
 children [
      Shape {
       appearance Appearance {
            material DEF M_L Material { diffuseColor 0 0.7 0.7}
       }
       geometry Text {
```

```
                string  "L"
                fontStyle FontStyle {
                size 3
                justify "MIDDLE"
                style "BOLDITALIC"
                family "VERDANA"
                }
        }
    }
]
}

Transform {
 translation -2.0 0.7 5
 children [
      Shape {
       appearance Appearance {
            material DEF M_C Material { diffuseColor 0.7 0.7 0 }
       }
       geometry Text {
                string  "C"
                fontStyle FontStyle {
                size 3
                justify "MIDDLE"
                style "BOLDITALIC"
                family "VERDANA"
                }
        }
    }
]
}

Transform {
 translation -0.3 0.9 2
 children [
      Shape {
       appearance Appearance {
            material DEF M_O Material { diffuseColor 0.0 0.0 0.1 }
       }
       geometry Text {
                string  "O"
                fontStyle FontStyle {
                size 3
                justify "MIDDLE"
                style "BOLDITALIC"
                family "VERDANA"
                }
        }
    }
 ]
}

Transform {
 translation 2.0 1.1 -1.0
 children [
      Shape {
       appearance Appearance {
            material DEF M_M Material { diffuseColor 0.1 0.7 0 }
       }
       geometry Text {
                  string "M"
                  fontStyle FontStyle {
                  size 3
                  justify "MIDDLE"
                  style "BOLDITALIC"
                  family "VERDANA"
                  }
         }
    }
 ]
}

Transform {
 translation 4.3 1.3 -3.5
 children [
      Shape {
       appearance Appearance {
            material DEF M_E2 Material { diffuseColor 0.1 0.7 0 }
       }
       geometry Text {
                  string "E"
                  fontStyle FontStyle {
                  size 3
                  justify "MIDDLE"
                  style "BOLDITALIC"
                  family "VERDANA"
                  }
         }
    }
 ]
}
```

```
Transform {
 translation 0 -3.6 0
 children [
       Shape {
        appearance Appearance {
        material Material { diffuseColor 0.3 0.3 0.8 emissiveColor 0.3 0.3 0.8  }
           }
        geometry DEF T_Text Text {
                    string "Solar System"
                    fontStyle FontStyle {
                     size 3.5
                     justify "MIDDLE"
                     style "BOLDITALIC"
                     family "ROCKWELL"
                    }
           }
       }
 ]
}


Transform {
 translation 0.25 -3.8 0
 children [
       Shape {
        appearance DEF A_Black Appearance {
             material Material { diffuseColor 0 0 0 }
        }
        geometry USE T_Text
       }
 ]
}


DEF TS_Neon TimeSensor {
 cycleInterval 1
 loop TRUE
}

DEF S_Neon Script {
 eventIn  SFTime   neonsign
 eventOut MFString sign
 field SFNode w USE M_W
 field SFNode e1 USE M_E1
 field SFNode l USE M_L
field SFNode c USE M_C
field SFNode o USE M_O
field SFNode m USE M_M
field SFNode e2 USE M_E2
field SFInt32 i 0
url "javascript:
        function initialize() {
                sign[0] =  '0000000';
                sign[1] =  '1000000';
                sign[2] =  '0100000';
                sign[3] =  '0010000';
                sign[4] =  '0001000';
                sign[5] =  '0000100';
                sign[6] =  '0000010';
                sign[7] =  '0000001';
                sign[8] =  '0000000';
                sign[9] =  '0000001';
                sign[10] = '0000010';
                sign[11] = '0000100';
                sign[12] = '0001000';
                sign[13] = '0010000';
                sign[14] = '0100000';
                sign[15] = '1000000';
                sign[16] = '0000000';
                sign[17] = '1111111';
                sign[18] = '0000000';
                sign[19] = '1111111';

                i = 0;
        }
        function neonsign() {
                if (sign[i].substring(0,1)=='0') {
                        w.diffuseColor = new SFColor(0.7,0,0);
                }
                else {
                        w.diffuseColor = new SFColor(1,0,0);
                }
                if (sign[i].substring(1,2)=='0') {
                        e1.diffuseColor = new SFColor(0,0.7,0);
                }
                else {
                        e1.diffuseColor = new SFColor(0,1,0);
                }
                if (sign[i].substring(2,3)=='0') {
                        l.diffuseColor = new SFColor(0,0.7,0.7);
                }
                else {
                        l.diffuseColor = new SFColor(0,1,1);
                }
                if (sign[i].substring(3,4)=='0') {
                        c.diffuseColor = new SFColor(0.7,0.7,0);
                }
                else {
                        c.diffuseColor = new SFColor(1,1,0);
                }
                if (sign[i].substring(4,5)=='0') {
                        o.diffuseColor = new SFColor(0.0,0,0.5);
                }
                else {
                        o.diffuseColor = new SFColor(0,0,1);
                }
                if (sign[i].substring(5,6)=='0') {
                        m.diffuseColor = new SFColor(0.7,0,0);
                }
                else {
                        m.diffuseColor = new SFColor(1,0,0);
                }
                if (sign[i].substring(6,7)=='0') {
                        e2.diffuseColor = new SFColor(0,0.7,0);
                }
                else {
                        e2.diffuseColor = new SFColor(0,1,0);
                }
                i++;
                if (i==20) {
                        i = 0;
                }
        }
 "
}

ROUTE TS_Neon.cycleTime TO S_Neon.neonsign

ROUTE TS_Ani.touchTime TO SoundWelcome.startTime
ROUTE TS_Ani.isOver TO TS_Time.enabled

ROUTE TS_Time.fraction_changed TO OI_Back.set_fraction
ROUTE OI_Back.value_changed TO T_Back.rotation
```

SOURCE CODE FOR THE VSS ANIMATION PAGE

```
#VRML V2.0 utf8

Group {
   children [
   # Stationary Sun
      Shape {
         appearance Appearance {
                     material Material {
               emissiveColor 1.0 0.3 0.1
                  }
                     texture ImageTexture{
                           url "sun.jpg"
                     }
                  }
         geometry Sphere { radius 2 }
      }
         # Generic lighting for ambience
      DirectionalLight {
         intensity 0.2
            ambientIntensity 1.0
      },
   # Several orbiting planets
   # The nearest to the most far away from the Sun
      DEF Mercury Transform {
         translation 3.0 0.0 0.0
         center -3.0 0.0 0.0
         children Shape {
            appearance Appearance {
                     material Material { diffuseColor 0 1 0}
                     texture ImageTexture{
                           url "mercury.jpg"
                     }
                  }
            geometry Sphere { radius 0.1 }
         }
      },
      DEF Venus Transform {
         translation 4.0 0.0 0.0
         center -4.0 0.0 0.0
         children Shape {
            appearance Appearance {
                     material Material { diffuseColor 0 0 1}
                     texture ImageTexture{
```

```
                    url "venus.jpg"
                }
            }
        geometry Sphere { radius 0.26 }
    }
 },
DEF Earth Transform {
        rotation 0.0 1.0 0.0 1.59
    translation 5.0 0.0 0.0
    center -5.0 0.0 0.0
    children Shape {
        appearance Appearance {
                material Material { diffuseColor 1 1 0}
                texture ImageTexture{
                    url "earth.jpg"
                }
            }
        geometry Sphere { radius 0.275 }
    }
},
    DEF Mars Transform {
    translation 6.0 0.0 0.0
    center -6.0 0.0 0.0
    children Shape {
        appearance Appearance {
                material Material { diffuseColor 1 0.5 0.2}
                texture ImageTexture{
                    url "mars.jpg"
                }
            }
        geometry Sphere { radius 0.15 }
    }
},
    DEF Jupiter Transform {
    translation 7.0 0.0 0.0
    center -7.0 0.0 0.0
    children Shape {
        appearance Appearance {
                material Material { diffuseColor 0.8 0.3 0.5}
                texture ImageTexture {
        url "jupiter.jpg"
        }
            }
        geometry Sphere { radius 1 }
    }
},
    DEF Saturn Transform {
    translation 8.0 0.0 0.0
    center -9.0 0.0 0.0
    children Shape {
      appearance Appearance {
             material Material { diffuseColor 0.5 0.1 0.7}
             texture ImageTexture{
                  url "saturn.jpg"
             }
          }
      geometry Sphere { radius 0.85 }
    }
  },
    DEF Uranus Transform {
    translation 9.0 0.0 0.0
    center -10.0 0.0 0.0
    children Shape {
      appearance Appearance {
             material Material { diffuseColor 0.5 0.6 0.4}
             texture ImageTexture{
                  url "uranus.jpg"
             }
          }
      geometry Sphere { radius 0.5 }
    }
  },
    DEF Neptune Transform {
    translation 10.0 0.0 0.0
    center -11.0 0.0 0.0
    children Shape {
      appearance Appearance {
             material Material { diffuseColor 0.2 0.1 0.3}
             texture ImageTexture{
                  url "neptunes.jpg"
             }
          }
      geometry Sphere { radius 0.575 }
    }
  },
    DEF Pluto Transform {
    translation 11.0 0.0 0.0
    center -12.0 0.0 0.0
    children Shape {
      appearance Appearance {
             material Material { diffuseColor 0.5 0.6 0.8}
             texture ImageTexture{
                    url "pluto.jpg"
                }
            }
        geometry Sphere { radius 0.05 }
      }
    },
# Animation clocks, one per planet
    DEF Clock1 TimeSensor {
      cycleInterval 10.7
      loop TRUE
    },
    DEF Clock2 TimeSensor {
      cycleInterval 20.3
      loop TRUE
    },
    DEF Clock3 TimeSensor {
      cycleInterval 30.1
      loop TRUE
    },
    DEF Clock4 TimeSensor {
      cycleInterval 40.1
      loop TRUE
    },
    DEF Clock5 TimeSensor {
      cycleInterval 50.1
      loop TRUE
    },
    DEF Clock6 TimeSensor {
      cycleInterval 60.1
      loop TRUE
    },
    DEF Clock7 TimeSensor {
      cycleInterval 70.1
      loop TRUE
    },
    DEF Clock8 TimeSensor {
      cycleInterval 80.1
      loop TRUE
    },
    DEF Clock9 TimeSensor {
      cycleInterval 90.1
      loop TRUE
    },
# Animation paths, one per planet
    DEF PlanetMercury OrientationInterpolator {
      key [ 0.0, 0.50, 1.0 ]
```

```
   keyValue [
      0.0 1.0 1.0  0.0,
      0.0 1.0 1.0  3.14,
      0.0 1.0 1.0  6.28
   ]
},
DEF PlanetVenus OrientationInterpolator {
   key [ 0.0, 0.50, 1.0 ]
   keyValue [
      0.0 1.0 1.0  0.0,
      0.0 1.0 1.0  3.14,
      0.0 1.0 1.0  6.28
   ]
},
DEF PlanetEarth OrientationInterpolator {
   key [ 0.0, 0.50, 1.0 ]
   keyValue [
      0.0 0.0 1.0  0.0,
      0.0 0.0 1.0  3.14,
      0.0 0.0 1.0  6.28
   ]
},
DEF PlanetMars OrientationInterpolator {
   key [ 0.0, 0.50, 1.0 ]
   keyValue [
      0.0 1.0 1.0  0.0,
      0.0 1.0 1.0  3.14,
      0.0 1.0 1.0  6.28
   ]
},
DEF PlanetJupiter OrientationInterpolator {
   key [ 0.0, 0.50, 1.0 ]
   keyValue [
      0.0 1.0 1.0  0.0,
      0.0 1.0 1.0  3.14,
      0.0 1.0 1.0  6.28
   ]
},
DEF PlanetSaturn OrientationInterpolator {
   key [ 0.0, 0.50, 1.0 ]
   keyValue [
      0.0 1.0 1.0  0.0,
      0.0 1.0 1.0  3.14,
      0.0 1.0 1.0  6.28
   ]
},
      DEF PlanetUranus OrientationInterpolator {
         key [ 0.0, 0.50, 1.0 ]
         keyValue [
            0.0 1.0 1.0  0.0,
            0.0 1.0 1.0  3.14,
            0.0 1.0 1.0  6.28
         ]
      },
      DEF PlanetNeptune OrientationInterpolator {
         key [ 0.0, 0.50, 1.0 ]
         keyValue [
            0.0 1.0 1.0  0.0,
            0.0 1.0 1.0  3.14,
            0.0 1.0 1.0  6.28
         ]
      },
      DEF PlanetPluto OrientationInterpolator {
         key [ 0.0, 0.50, 1.0 ]
         keyValue [
            0.0 1.0 1.0  0.0,
            0.0 1.0 1.0  3.14,
            0.0 1.0 1.0  6.28
         ]
      },
   ]
}


   # Light bulb
      DEF BulbLight PointLight {
         radius 16.0
         color 1.0 0.7 0.0
      },



   # Animation clock
      DEF Clock TimeSensor {
         cycleInterval 4.0
         loop TRUE
      },


   # Animation brightness and colors
```

```
DEF BulbIntensity ScalarInterpolator {
   key [ 0.0, 0.5, 0.5, 1.0 ]
   keyValue [ 1.0, 1.0, 0.0, 0.0 ]
},
DEF BulbDiffuse ColorInterpolator {
   key [ 0.0, 0.5, 0.5, 1.0 ]
   keyValue [
      0.0 0.0 0.0,  0.0 0.0 0.0,
      1.0 0.3 0.3,  1.0 0.3 0.3
   ]
},
DEF BulbEmissive ColorInterpolator {
   key [ 0.0, 0.5, 0.5, 1.0 ]
   keyValue [
      1.0 0.3 0.3,  1.0 0.3 0.3,
      0.0 0.0 0.0,  0.0 0.0 0.0,
   ]
},


   Anchor {
   children [
# Rotating Cube
DEF Cube1 Transform {
      translation 4.3 -1.8 5
   children Shape {
      appearance Appearance {
         material Material { diffuseColor 1 1 0}
      }
      geometry Box { size 0.5 0.5 0.01 }


   }
},


   DEF Text1 Transform {
      translation 4.287 -1.87 5.01
   children Shape {
      appearance Appearance {
         material Material { diffuseColor 0.25 0.25 0.25}
      }
         geometry Text {
         string "Stop"
         fontStyle FontStyle {
                  family "CHIANTI BT"
```

```
                    size 0.25
                    style "BOLD"
               justify "MIDDLE"
             }
           }

       }
     },
     ]
       description "stop animation"
       url "orbitingplanet2.wrl"
       }


ROUTE Clock1.fraction_changed   TO PlanetMercury.set_fraction
ROUTE Clock2.fraction_changed   TO PlanetVenus.set_fraction
ROUTE Clock3.fraction_changed   TO PlanetEarth.set_fraction
ROUTE Clock4.fraction_changed   TO PlanetMars.set_fraction
ROUTE Clock5.fraction_changed   TO PlanetJupiter.set_fraction
ROUTE Clock6.fraction_changed   TO PlanetSaturn.set_fraction
ROUTE Clock7.fraction_changed   TO PlanetUranus.set_fraction
ROUTE Clock8.fraction_changed   TO PlanetNeptune.set_fraction
ROUTE Clock9.fraction_changed   TO PlanetPluto.set_fraction
ROUTE PlanetMercury.value_changed TO Mercury.set_rotation
ROUTE PlanetVenus.value_changed TO Venus.set_rotation
ROUTE PlanetEarth.value_changed TO Earth.set_rotation
ROUTE PlanetMars.value_changed TO Mars.set_rotation
ROUTE PlanetJupiter.value_changed TO Jupiter.set_rotation
ROUTE PlanetSaturn.value_changed TO Saturn.set_rotation
ROUTE PlanetUranus.value_changed TO Uranus.set_rotation
ROUTE PlanetNeptune.value_changed TO Neptune.set_rotation
ROUTE PlanetPluto.value_changed TO Pluto.set_rotation
```

APPENDIX C

USER MANUAL

CHAPTER 1 INTRODUCTION

The Virtual Solar System (VSS) is a Virtual Reality application that allows user to navigate in the VR world. The Graphical User Interface (GUI) provides an easy to learn and user friendly environment. This user menu contains step-by-step guides for beginner on how to use the VSS.

1.1 HARDWARE REQUIREMENTS

The hardware configuration requirements for accessing VSS are as follows:

- Pentium 250MHz processor
- 32MB RAM
- 32X CD-ROM drive
- 14" Colour SVGA Monitor (with resolution at 800 X 600)
- Mouse
- 101 Enhance Keyboard

1.2 SOFTWARE REQUIREMENT

VSS requires the following software as its running platform:

- Operating system:

Microsoft Windows 95/98/NT and above.

- Web browser:

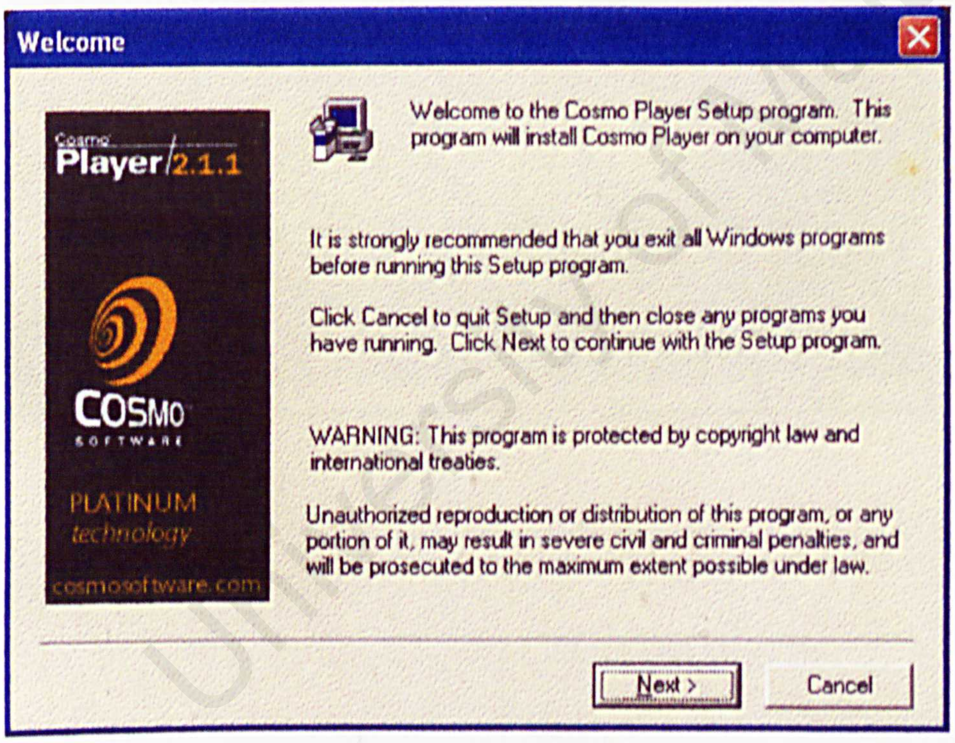
Microsoft Internet Explorer 5.5 and above or Netscape Communicator 4.5 and above.

- Cosmo Player 2.1 (provided in the CD)

1.3 HOW TO INSTAL COSMO PLAYER 2.1

In order to run the application successfully, the above software must be installed first by using the following steps:

- Double click the Cosmo Player icon in the CD
- A *Welcome* page will be displayed as shown below and click NEXT to continue.
- Select YES to accept the *Software License Agreement* on the following page.
- Choose the defaulted option at the *Select Components* and click NEXT.
- Use the defaulted option at *Choose Destination Location* and select NEXT.
- At the *Setup Complete*, click FINISH to complete the installation.



Welcome to the Cosmo Player Setup program. This program will install Cosmo Player on your computer.

It is strongly recommended that you exit all Windows programs before running this Setup program.

Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

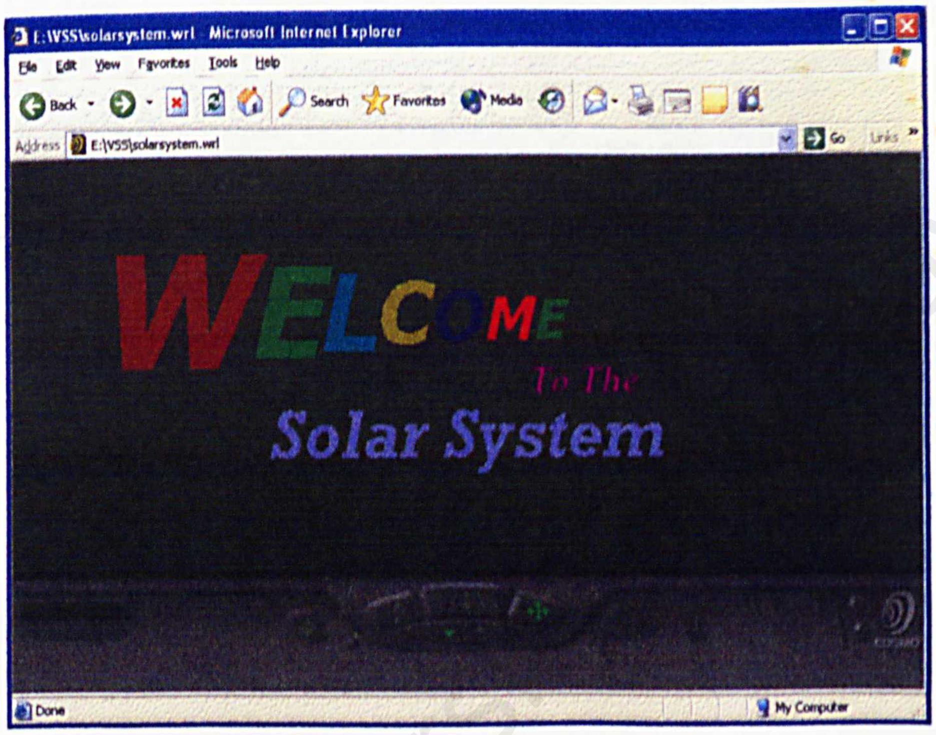
WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

1.4 HOW TO RUN VSS

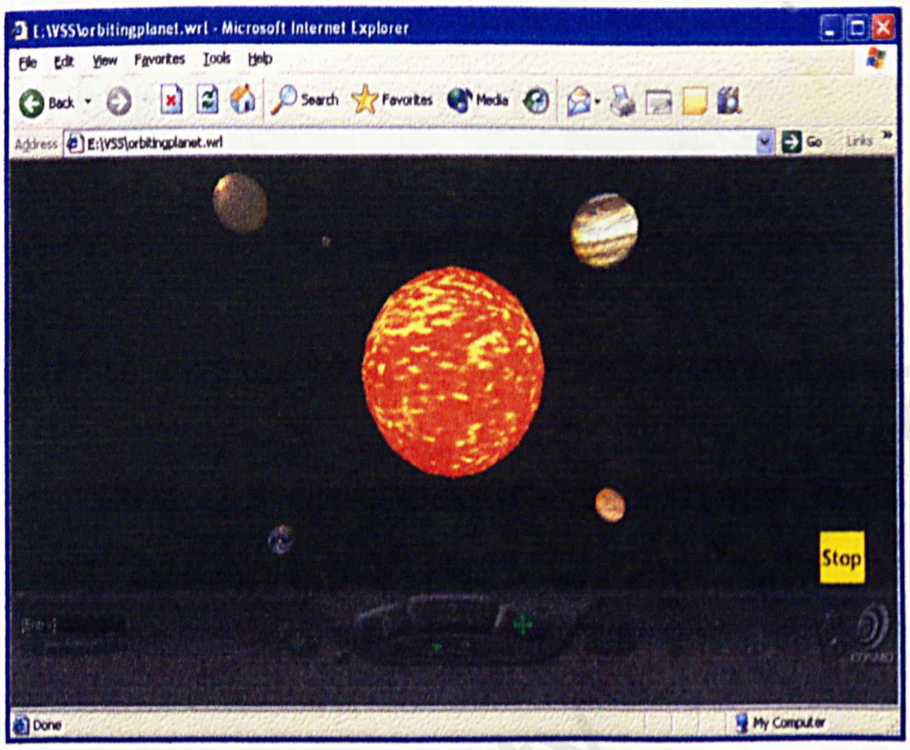
1.41 SCREEN 1

- In the CD, open the *VSS* folder and click on the *solarsystem* icon.
- A welcome page will be displayed as shown below.
- Click on the screen and it will lead you to the page.


E:\VSS\solarsystem.wrl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\VSS\solarsystem.wrl
Go Links
WELCOME
To The
Solar System
Done
My Computer

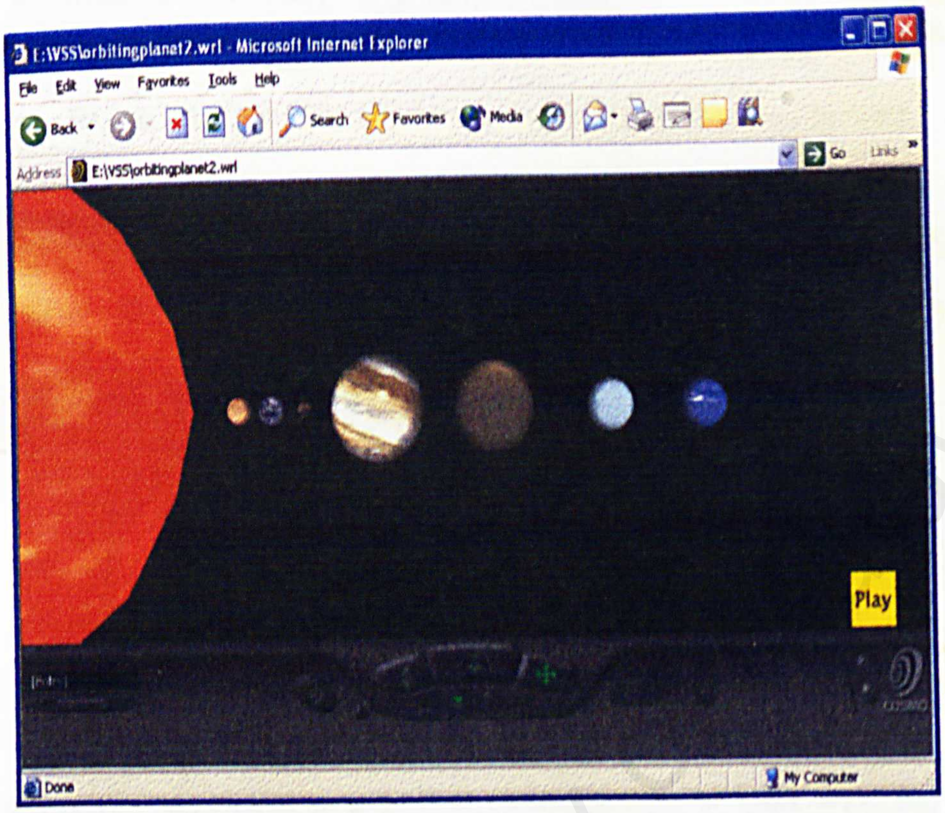
1.42 SCREEN 2

- You will notice that all the planets revolving around the Sun.
- Select the STOP button to stop the animation.
- It will lead you to another page.


E:\VSS\orbitingplanet.wrl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\VSS\orbitingplanet.wrl
Go Links
Stop
Done
My Computer

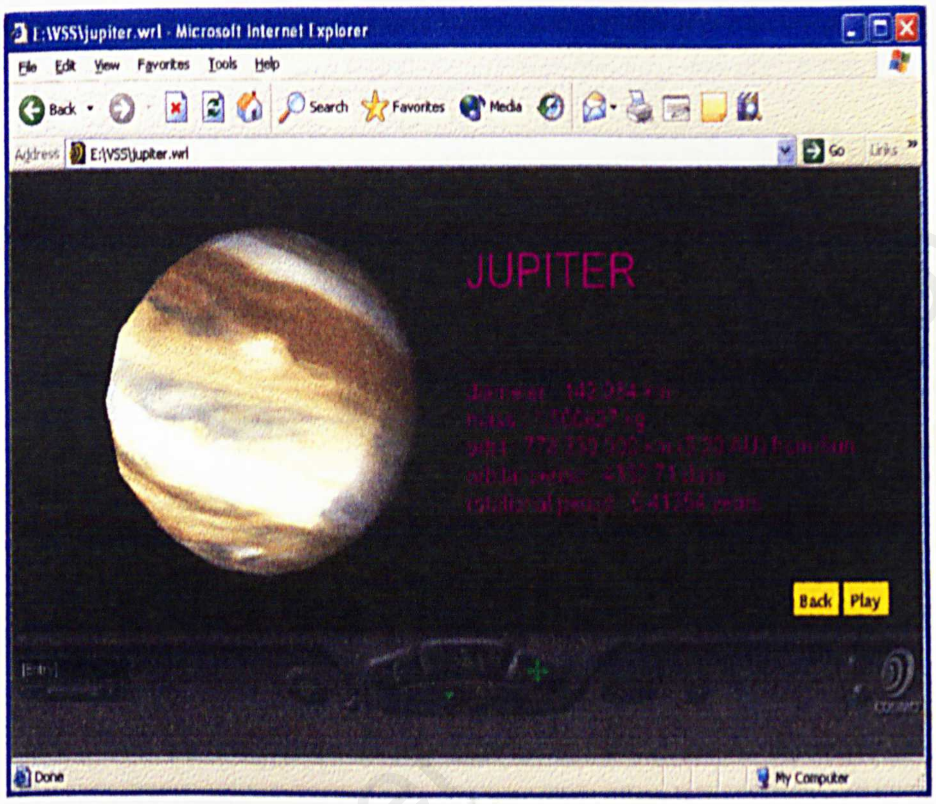
1.43 SCREEN 3

- Click on any planets to view the information on next page.
- Select the PLAY button to go back to Screen 2.


E:\VSS\orbitingplanet2.wrl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\VSS\orbitingplanet2.wrl
Go Links
Play
Done
My Computer

1.44 SCREEN 4

- Click on the planet to listen to its pronunciation.
- Select BACK button to go back to Screen 3.
- Click on PLAY button to go back to Screen 2.
- To find out more about Cosmo Player, select the "?" button.
- To exit VSS, just close off the Microsoft Internet Explorer browser.


E:\VSS\jupiter.wrl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\VSS\jupiter.wrl
Go Links
Back
Play
Done
My Computer

JUPITER

diameter: 142,984 km
mass: 1.900e27 kg
orbit: 778,330,000 km (5.20 AU) from Sun
orbital period: 4332.71 days
rotational period: 0.41354 years

REFERENCES

REFERENCES

1. Michele Matossian, 2001. 3ds Max 4 For Windows. Peachpit Press.
2. Gavin Bell, Anthony Parisi and Mark Pesce, The Virtual Reality Modeling Language, Version 1.0 Specification. The VRML Homepage, Wired.
3. Chris Marrin & Bruce Cambell, 1977. Teach Yourself VRML 2 in 21 days 1st ed., Indianapolis, Ind.Sama.net.
4. John Vacca, 1998. VRML Clearly Explained" 2nd ed., Boston: AP Professional.
5. Andrea L. Ames, David R. Nadeau & John L. Moreland, 1997. VRML 2.0 Sourcebook, 2nd ed. John Wiley & Sons, Inc.
6. Roger Pressman, 1992. Software Engineering A Practitioner's Approach. McGraw-Hill Inc.
7. http://www.solarviews.com/eng/homepage.htm
8. http://seds.lpl.arizona.edu/nineplanets/nineplanets/nineplanets.html
9. http://pads.jpl.nasa.gov/planets