P-1595

VIRTUAL 3D GAME DEVELOPMENT

A PROJECT REPORT

Submitted by

SUNDARAM.RM [71202104046]

ARUN KUMAR.B [71202104055]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE

KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE-641006

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2006

ANNAUNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"Virtual 3D Game Development"** is the bonafide work of **"Sundaram.RM** and **Arun Kumar.B"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE** 26/04/06
THANGASWAMY.S	SIVAN ARUL SELVAN.K
HEAD OF THE DEPARTMENT	**SUPERVISOR**
	SENIOR LECTURER
Department of Computer Science & Engineering	Department of Computer Science & Engineering
Kumaraguru College of Technology	Kumaraguru College of Technology
Chinnavedampatti	Chinnavedampatti
Coimbatore – 641 006	Coimbatore – 641 006

Submitted for the viva voce examination held on 2/5/2K6

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We hereby declare that the project entitled **"VIRTUAL 3D GAME DEVELOPMENT"** is a record of original work done by us and to the best of our knowledge; a similar work has not been submitted to Anna University or any institution, for fulfilment of the requirement of the course study.

The report is submitted in partial fulfilment for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore
Date: 26/4/06

[Sundaram.RM]

[Arun Kumar.B]

ACKNOWLEDGEMENT

With profound gratitude, we express our deepest thanks to our internal guide Mr. K. Sivan Arul Selvan, Senior Lecturer, Department of Computer Science and Engineering, who has taken all measures to guide us through the project, and been a constant source of inspiration and motivation at various levels of the project.

Our sincere thanks to all the lab technicians who have been operational in aiding us implement the system.

We would like to thank the Head of the Department, Computer Science and Engineering, Dr. S. Thangasamy and Mrs. P. Devaki, Project Coordinator for guiding us through the project.

Our sincere thanks to the Department of Computer Science and Engineering, Kumaraguru College of Technology, for extending its fullest support by all means that enabled us to complete the project in a timely manner.

We extend our utmost gratitude to our friend Mr. C.S. Shyam Sundar who gave us the idea and avenue to do such a project in gaming. Last but not the least; thanks to our parents, friends, and all those who directly or indirectly helped us in successful completion of the project.

ABSTRACT

The objective of this project is to create a 3D game (First Person Shooter), which has an interactive mixture of Audio and Video output based on the actions performed by the player.

The project focuses on creating a virtual world that provides the user (player) with an animated scenario on which inputs can be made as actions and outputs can be visualized reactions in the virtual world.

The system displays the virtual world using 3D models that are modelled, textured (to provide realism) and rendered in an interrelated manner so as to simulate a virtual world. The rendering is finally done with the use of Direct X (a Microsoft graphics library which provides C API's using which one can manipulate the Audio & Video of a system, without getting into the hardware level intricacies).

3D Studio Max is used for the modelling and texturing of the objects used in the virtual world with the use of various modelling techniques.

DarkBASIC, a game programming language, with BASIC language's syntax, provides an interface to the DirectX 'C' APIs. This is used for the programming where the complete synchronisation is done.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS & SYMBOLS

ABBREVATION	DESCRIPTION
3D	Three Dimensional
2D	Two Dimensional
3ds Max	3D Studio Max
Max	3D Studio Max
BLIT	Bit Block Transfer
NURMS	Non-Uniform Rational Mesh Smooth
NURBS	Non-Uniformal Rational B-Spline
B – spline	Bézier spline curve
PArray	Particle Array
PCloud	Particle Cloud
API	Application Programming Interface
SDK	Software Development Kit
COM	Component Object Model
D3D	Direct3D
DS3D	DirectSound3D
REF	Reference Rasterizer
Win32	32 bit Windows
FTP	File Transfer Protocol
AVI	Audio Video Interleaved
DLL	Direct Link Library
LAN	Local Area Network
FPS	First Person Shooter

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Games are something that have always fascinated and attracted the young and the old equally. This project titled **Virtual 3D Game Development** is a 3D game that is designed to enthral different varieties of players.

This virtual 3D game named Dark Rider is a typical first person shooter (camera view is shown from the hero character's perspective view) game. It posses all the necessary features and gaming experience a 3D game offers.

The project is said to be completed in four phases. Namely:

- **Modelling**

  This phase involves the creation of all the visual components of the game; this is done using 3D Studio Max. Each of the visual components should be created separately so that the load can be minimized. Each and every object needed to be shown on screen is to be modelled separately using this tool, so that it can be loaded before game play.

- **Texturing**

  Modelling creates the objects physical shape alone. To get the realism and the real world feel for objects, they are to be textured.

- **Animation**

  Once the objects are created and textured, they are ready for actions that make them act real. This is done by animating the objects as needed in the game. Different actions and expression are pre-animated and embedded within the object. When needed, the particular animation sequence within is called.

- **Synchronisation**

  This is the stage where all the above done phases show out their functionality. All objects are loaded and placed in the required place within the created virtual world. As per the player's input the animations are run and the sound effects are played.

1.2 THE BASIC ENGINE

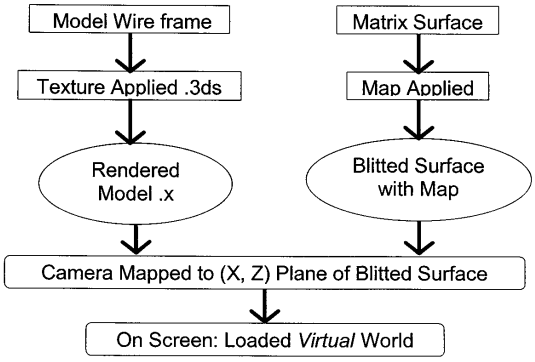

Fig 1.1: The Basic Engine

MODELLING METHODS IN 3D STUDIO MAX

CHAPTER 2

MODELLING METHODS IN 3D STUDIO MAX

2.1 OVERVIEW

3D Studio Max is a three dimensional vector graphics and animation software, by Autodesk Media & Entertainment.

3ds Max is one of the most widely used 3D animation suite. It has strong modelling and texturing capabilities, a ubiquitous plug-in architecture and a long heritage on the Microsoft Windows platform. *3ds Max* is mostly used by video game developers but can also be used for pre-rendered productions such as movies, special effects and architectural presentations.

2.2 MODELLING METHODS

3ds Max offers a wide range of modelling techniques so as to model almost all real world objects and organisms. There are 5 basic modelling methods. They are as follows:

- Modelling with primitives
- NURMS (*N*on-*U*niform *R*ational *M*esh *S*mooth)
- Surface tool/Editable patch object
- NURBS (*N*on-*U*niformal *R*ational *B*-*S*pline)
- Polygon modelling

2.2.1 MODELLING WITH PRIMITIVES

This is a basic method, in which the modelling of a complex object is done only using boxes, spheres, cones, cylinders and other predefined objects. One may also apply boolean operations, including subtract, cut and connect. For example, one can make two spheres which will work as blobs that will connect with each other. This is called "blob-mesh modelling," or "balloon modelling."

STANDARD PRIMITIVES

The Standard Primitives are: Box, Sphere, Cylinder, Torus, Teapot, Cone, Geosphere, Tube, Pyramid, and Plane.

2.2.1.1 BOX

In geometry, a **cuboid** is a solid figure bounded by six rectangular faces: a **rectangular box**. All angles are right angles, and opposite faces of a cuboid are equal. It is also a **right rectangular prism**. This primitive model is subdivided into segments of definite length and breadth. The number of segments defines the model's smoothness.

If the dimensions of a cuboid are *a*, *b* and *c* then its volume is *abc* and its surface area is $2ab + 2bc + 2ac$.
The length of the space diagonal is $d = \sqrt{a^2 + b^2 + c^2}$.

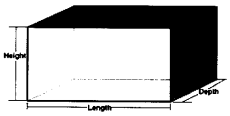

Fig 2.1: The Box Primitive

2.2.1.2 SPHERE

A **sphere** is a perfectly symmetrical geometrical object. In mathematics, the term refers to the surface or boundary of a **ball**, but in non-mathematical usage, the term is used to refer either to a three-dimensional ball or to its surface. It is heavily used in almost all complex models.

If *r* is the radius of the sphere then its volume is $4/3\,\pi r^3$ and the surface area is $4\,\pi r^2$. The diameter is 2r.

The sphere model is made of a number of interconnected arcs. This model has the number of segments parameter to define its smoothness.

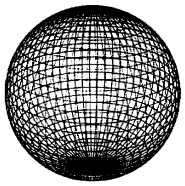

Fig 2.2: Sphere Primitive – Wire frame

2.2.1.3 CYLINDER

A right circular cylinder, in mathematics, a cylinder is a quadric, i.e. a three-dimensional surface, with the following equation in Cartesian coordinates:

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

This equation is for an elliptic cylinder, a generalization of the ordinary, circular cylinder (a = b). Even more general is the generalized cylinder: the cross-section can be any curve.

If the cylinder has a radius r and length h, then its volume is given by $V = \pi r^2 h$ and its surface area is $A = 2\pi r(r + h)$

The cylinder model is made of fine spherical discs of definite radius. The number of discs defines the curvature of the model.

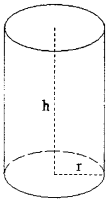

Fig 2.3: The Cylinder Primitive

2.2.1.4 TORUS

In geometry, a torus (pl. tori) is a doughnut-shaped surface of revolution generated by revolving a circle about an axis coplanar with the circle. The sphere is a special case of the torus obtained when the axis of rotation is a diameter of the circle. If the axis of rotation does not intersect the circle, the torus has a hole in the middle and resembles a ring doughnut, a hula-hoop or an inflated tire. The other case, when the axis of rotation is a chord of the circle, produces a sort of squashed sphere resembling a round cushion.

This model is made of a number of interconnected circles. The vertical ones are for curvature and horizontal ones for bulging dimension.

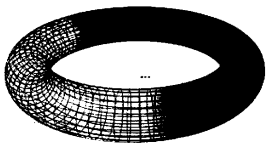

Fig 2.4: The Torus Primitive

A torus can be defined parametrically by:
$x(u, v) = (R + r\cos v)\cos u$
$y(u, v) = (R + r\cos v)\sin u$
$z(u, v) = r\sin v$
where
$u, v \in [0, 2\pi]$,
R is the distance from the centre of the tube to the centre of the torus,
r is the radius of the tube.

2.2.1.5 CONE

In common usage and elementary geometry, a **cone** is a solid object obtained by rotating a right triangle around one of its two short sides, the cone's **axis**. The disk swept by the other short side is called the **base**, and the endpoint of the axis which is not on the base is the cone's **apex** or **vertex**. An object shaped like a cone is said to be **conical**.

The volume, *V*, of a cone of height, *h*, and base radius, *r*, is 1/3 of the volume of the cylinder with the same dimensions, i.e. $V = \pi r^2 h / 3$. The centre of mass (assuming the cone is filled with uniform density) is located on the axis, 1/4 of the way from base to apex.

EXTENDED PRIMITIVES

The Extended Primitives available in 3ds Max include Hedra, ChamferBox, OilTank, Spindle, Gengon, Prism, Torus knot, ChamferCyl, Capsule, L-Ext, C-Ext and Hose.

2.2.2 NURMS

Non-uniform rational mesh smooth (*NURMS*) or subdivision surface technique uses a low-polygonal mesh to control the shape of the smooth surface. This is the *3ds Max* implementation of subdivision surface modelling, a methodology that is rapidly displacing NURBS modelling as the methodology of choice for both low- and high-polygon modelling. Max implements NURMS as a modifier, to be applied to a polygon or mesh object. After creating a rough-edged model (including all the desired detail) with polygons, Meshsmooth (the modifier containing NURMS) is applied. An advantage of NURMS Meshsmooth is that every vertex and edge has its own weight. The numeric value, weight, determines how strongly each vertex/edge will influence the final shape.

2.2.3 SURFACE TOOL

Surface tool is for creating common 3DS Max's splines, and then applying a modifier called "surface." This modifier makes a surface from every 3 or 4 vertices in a grid. This is often seen as an alternative to 'Mesh' or 'Nurbs' modelling, as it enables a user to interpolate curved sections with straight geometry (for example a hole through a box shape). Although the surface tool is a useful way to generate parametrically accurate geometry, it lacks the 'surface properties' found in the similar Edit Patch modifier, which enables a user to maintain the original parametric geometry whilst being able to adjust "smoothing groups" between faces.

2.2.4 NURBS

NURBS stands for *N*on-*U*niformal *R*ational *B*-Spline. A NURBS curve is defined by its order, a set of weighted control points, and a knot vector. NURBS curves and surfaces are generalizations of both B-splines and Bézier curves and surfaces, the primary difference being the weighting of the control points which makes NURBS curves rational (non-rational B-splines are a special case of rational B-splines). Whereas NURBS curves evolve into only one parametric direction, usually called s or u, NURBS surfaces evolve into two parametric directions, called *s* and *t* or *u* and *v*.

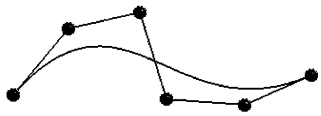

Fig 2.7: The Simple NURBS Curve

By evaluating a NURBS curve at various values of the parameter, the curve can be represented in Cartesian two- or three-dimensional space. Likewise, by evaluating a NURBS surface at various values of the two parameters, the surface can be represented in Cartesian space.

The statement that NURBS curves are a generalization of Bézier curves means that all Bézier curves are NURBS curves, but not all NURBS curves are Bézier curves.

This type is primarily useful for modelling indefinite shape objects such as plants, trees, hair, curved surfaces like mountains, slopes, etc.

2.2.5 POLYGON MODELLING

Polygon modelling is more common with game design than any other technique. Usually, the modeller begins with one of the 3ds max primitives, and using such tools as bevel, extrude, and polygon cut, adds detail to and refines the model. This technique is most often used for game design, as the very specific control over individual polygons allows for extreme optimization.

2.3 PARTICLE SYSTEMS

3ds max allows the designer to create an object called a particle emitter which allows the designer to treat all the particles of a particular type as a single group.

A particle emitter is an object in a 3D modelling program which emits objects that are treated as a group. As of version 7, there are 7 3ds max particle emitters. The 7 3DS Max particle emitters are PF Source, Spray, Snow, Blizzard, PArray, PCloud, and Super Spray. The most flexible of these particle systems is PF Source, which makes use of a technology called particle flow.

Fig 2.8: Galaxy & Fire Effect created using Particle Systems

CHAPTER 3

DIRECTX LIBRARIES AND DARKBASIC

3.1 DIRECTX OVERVIEW

DirectX is a collection of APIs for easily handling tasks related to game programming on the Microsoft Windows operating system. It is most widely used in the development of computer games for Microsoft Windows. The DirectX SDK is available free from Microsoft. The DirectX runtime was originally redistributed by computer game developers along with their games, but later it was included in Windows. DirectX 9.0c is the latest release version of DirectX. DirectX 10 Beta is available as of Windows Vista build 5238. The latest versions of DirectX are still usually included with PC games, since the API is updated so often. It is this library that synchronises all the audio and video into a single system without letting the programmer to bother about the hardware intricacy.

3.2 DIRECTX APIs

The various components of DirectX are in the form of COM-compliant objects. The components comprising DirectX are:

- **DirectX Graphics**, comprised of two APIs (DirectX 8.0 onwards):
  - DirectDraw: for drawing raster graphics
  - Direct3D: for drawing 3D graphics primitives
- DirectInput: used to process data from a keyboard, mouse, joystick, or other game controllers
- DirectPlay: for networked communication of games

- DirectSound: for the playback and recording of waveform sound
  - DirectSound3D: for the playback of 3D sounds.
- DirectMusic: for playback of soundtracks authored in DirectMusic Producer
- DirectSetup: for the installation of DirectX components
- DirectX Media: comprising of DirectAnimation, DirectShow and DirectX Transform for animation, media streaming applications, and interactivity respectively
- DirectX Media Objects: support for streaming objects such as encoders, decoder and effects

3.2.1 DIRECTDRAW

DirectDraw is used to render graphics in applications where top performance is important. DirectDraw also allows applications to run full screen instead of embedded in a window such as most other Windows applications. DirectDraw uses hardware acceleration if it is available on the client's computer.

DirectDraw is a 2D API. That is, it contains commands for 2D rendering and does not support 3D hardware acceleration. A programmer could use DirectDraw to draw 3D graphics, but the rendering would be slow compared to an API such as Direct3D which does support 3D hardware acceleration.

As of DirectX version 8.0, DirectDraw was no longer updated and available directly in DirectX. Some of DirectDraw's functionality was rolled over into a new package called DirectX Graphics, which was really just Direct3D with a few DirectDraw API additions.

3.2.2 DIRECT3D

Direct3D is used to render three dimensional graphics in applications where performance is important, such as games. Direct3D also allows applications to run full screen instead of embedded in a window, though they can still run in a window if programmed for that feature. Direct3D uses hardware acceleration if it is available on the graphic board.

Direct3D is a 3D API. That is, it contains many commands for 3D rendering, but contains few commands for rendering 2D graphics. Microsoft strives to continually update Direct3D to support the latest technology available on 3D graphics cards. Direct3D offers full vertex software emulation but no pixel software emulation for features that aren't available in hardware. For example, if a program programmed using Direct3D requires pixel shaders and the graphics card on the user's computer does not support that feature, Direct3D will not emulate it. The program will most likely exit with an error message. The API does define a *Reference Rasterizer* (or REF device), which emulates a generic graphics card, although it's too slow to be used in any application to emulate pixel shaders and is usually ignored.

3.2.3 DIRECTINPUT

Microsoft DirectInput is an application programming interface (API) for input devices including the mouse, keyboard, joystick, and other game controllers, as well as for force-feedback (input/output) devices.

Apart from providing services for devices not supported by the Microsoft Win32 API, DirectInput gives faster access to input data by

communicating directly with the hardware drivers rather than relying on Microsoft Windows messages.

DirectInput enables an application to retrieve data from input devices even when the application is in the background. It also provides full support for any type of input device, as well as for force feedback. Through action mapping, applications can retrieve input data without needing to know what kind of device is being used to generate it.

The extended services and improved performance of DirectInput make it a valuable tool for games, simulations, and other real-time interactive applications running under Windows.

3.2.4 DIRECTSOUND

It is a part of the DirectX library, supplied by Microsoft, which resides on a computer with the Windows operating system. It provides the interface between applications and the sound card, enabling applications to produce sounds and music. Besides providing the essential service of passing audio data to the sound card, it provides many needed capabilities. Of these audio mixing and volume control are the most essential. DirectSound also allows several applications to conveniently share access to the sound card at the same time. Its ability to play sound in 3D added a new dimension to games. It also provides the ability for games to modify a musical script in response to game events in real time, i.e. the beat of the music could quicken as the action heats up. DirectSound also provides effects such as echo, reverb, and flange. After many years of development, today DirectSound is a very mature API, and supplies many other useful capabilities, such as the ability to play multi-channel sound and sounds at high resolution.

3.3 DARKBASIC OVERVIEW

DarkBASIC Professional is one of the most advanced games development package built on the BASIC language currently available. No other package out there makes it as easy to incorporate all of the special features and effects you see in today's games and no other package natively offers the benefits of Microsoft's DirectX 9 technology. The sole purpose of the language was game creation using Microsoft's DirectX 3D library for graphics but is well rounded enough to do almost any programming task.

3.4 MAJOR COMMAND SETS

- BASIC Commands: Common commands shared with most other BASIC languages. Control loops, User functions, Condition statements, etc.
- Input Commands: Includes input functions for FTP, Files, and Joysticks in addition to mouse and keyboard inputs.
- Math Commands: Math functions, ranging from arithmetic to trigonometry.
- Basic 2D: As the header implies, basic 2D drawing functions such as lines and ellipses
- Text: Contains functions for printing text to the screen, and manipulating strings.
- Screen: Controls display modes, screen resolution, etc.
- Bitmap: Contains functions for bitmap manipulation such as fading and flipping.
- Sprite: Functions for image and sprite manipulation.
- Sound: Sound manipulation and 3D sound systems.

- Music: Loading and playing music.
- Animation: Manipulation of AVI files.
- Basic 3D: Controls 3D object position, rotation, limbs, and animation.
- Camera 3D: Controls 3D camera position, rotation, field of view, and effects.
- Light 3D: Manipulation of 3D lighting.
- Matrix 3D: Create and manipulate 3D matrices.
- System: Contains DLL, window, and hardware commands.
- Memblock: Memblock manipulation of images, sounds, meshes, and other data down to a per-byte level.
- Multiplayer: Commands for programs working across multiple machines over LAN or the internet, especially net-games.

MODELLING OBJECTS

CHAPTER 4

MODELLING OBJECTS

4.1 PENCIL SKETCH & REFERENCE IMAGE

Modelling is started with importing a reference images to right, and left view as background image. These images are very helpful in modelling process as a reference. When importing image as background "Fit Image" is chosen for the images to fit to the view port.

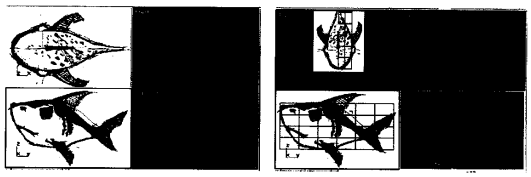

Fig 4.1: Fixing the Reference Image & Setting a Base

In right perspective view, draw a rectangular box that is not wider or higher than model to be created. Box has 6x4 segments (*l***b*) and 2 side segments (*depth*).

This box will be one half of the model. Maybe in top view the box isn't fitting the model. But do not move the box. Try to rescale the background image so it fits the box.

Once done, the background is locked, so when the box is moved, the image goes with it.

4.2 POSITIONG AND SHAPING THE BASE

The box is just one half of the Shark. This is useful in modelling practise because one can mirror this box as instance, and work only on one half of the head (box). And every time a change is made on one half it can be seen reflected on the other half too. Convert the box to an Editable Mesh by right-clicking and choosing 'Convert to Editable Mesh'.

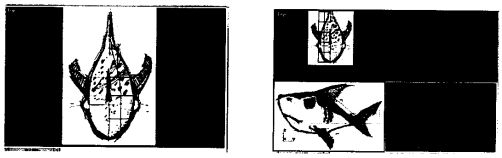

Fig 4.2: Positioning and Scaling the Base

Shape the vertices to match the reference images. You can see some big difference from the box. Most of work on this phase is moving those vertexes on the sharp edges of the box to make some smooth surfaces.

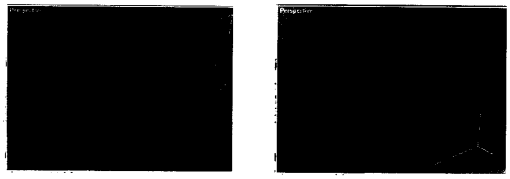

Fig 4.3: Moving and shaping the vertices

The model is now fine tuned by pulling and pushing the edges and vertices so as to get the desired level of bump or depression in the model.

4.3 APPLYING MESHSMOOTH

Once complete, apply the 'Meshsmooth Modifier' by right clicking and selecting the option. By turning off **show end result** button in Editable Mesh box, one can get to the previous step of modelling process, to fine tune it further.

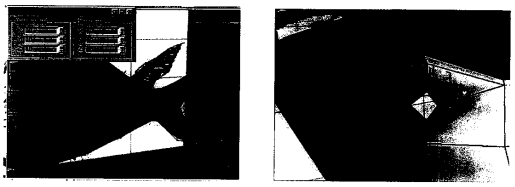

Fig 4.4: Applying Meshsmooth

The next step is to model a nose hole for the model. This is done by adding (cut tool) more edges. Delete faces where the hole is to be created. And extrude edges inside the model. Select all edges around that hole, press and hold shift and move them inside as shown in Fig 4.4 b.

Fig 4.5: Appending simple models to the base

Eye would be a sphere in this phase. The eye object is positioned and locked in the side view. Similar to the creation of nose hole, the eye socket is also created and a sphere is placed for eye ball.

4.4 FINISHING TOUCHES – FINE TUNING

Once the coarse model is complete, some final touches are made to make it look alive. Cutting and deleting of faces is done to reflect reality. These are done in place where greater smoothness is needed. Extruding edges around holes is done by selecting them and moving them from the model while pressing shift. This gives more curvature to holes.

Fig 4.6: Wire frame of a finished model

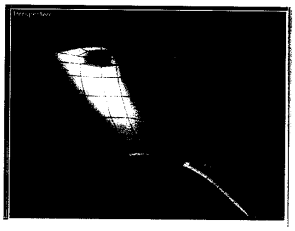

Fig 4.7: A finished model

TEXTURING MODELS

CHAPTER 5

TEXTURING MODELS

5.1 PRE – TEXTURING WORK

Before starting the process of texturing some preset settings are to be loaded. Only the editable mesh is kept for texturing. Then in Sub-object Polygon mode, 'All Polygons (faces)' is selected. All the faces are selected. In 'Modifier Stack', the Material ID is set 1. Now every face has number 1 material assigned.

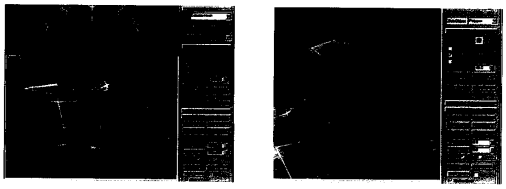

Fig 5.1: Pre-Texturing work and Sectioning

5.2 SECTIONING PARTS FOR TEXTURING

Polygons (faces) that represent different texturing parts are selected. Then they are given different material ID, instead of the old ID of the base. Now the base will be with a material 1 and other parts like back fin will have material 2 assigned, tail will have material ID 3 assigned.

5.3 TEXTURE MAPPING

All faces with a particular material ID that is to be textured is selected. While these faces are selected UVW Mapping modifier with Cylindrical mapping method is used. The cylindrical gizmo is put in the right direction; Sub-Object-Gizmo is used for selection. Click on rotate button with left mouse button and than with right mouse button to open this rotate menu. Rotate gizmo -90 around Z axes.

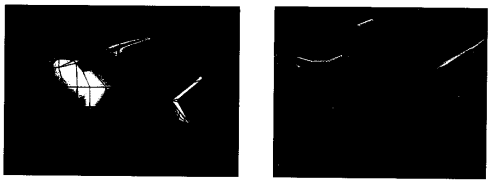

Fig 5.2: Mapping textures to different zones

The green line in Fig 5.2 (a) at the top of the gizmo denotes where mesh will get split when unwrapped. The same process is done to other zones of the model which have a different material ID.

When all parts are done with mapping in the Edit Modifier Stack 'Collapse All' method is applied. As a result the model is collapsed. Unwrap UVW modifier is applied. Then on pressing the Edit a new Edit UVW Window is opened. This window shows texture cylinder in unwrapped state which has all the material's unwrapped texture in an overlapped format Fig 5.3 (b).

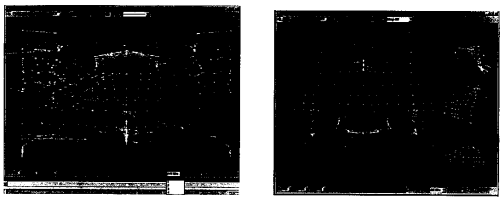

Fig 5.3: UVW Texturing & Segregation of Textures

After adjusting, moving, scaling one should have Fig 5.3 (b) in Edit UVW window. The main body is rotated 180 degrees, scaled down a little and mesh is much cleaner in the parts where it needs to be.

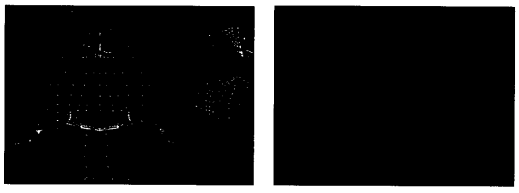

Fig 5.3: Texture Creation & Textured Model

When adjusting of unwrapped mesh is done, the image is taken to Photoshop. Print Screen is used to paste it in to Paint window. This image is pasted on to a separate layer. One layer is the image of mesh; other is dark parts of the back.

Once the painting is done, it is exported to 3ds Max and applied on to the model via the Material Editor, thus resulting in a fully textured model with a high degree of realism.

CHAPTER 6

THE GAME ENGINE
ANIMATION & SYNCHRONISATION

6.1 OVERVIEW

The models thus created by modelling and texturing are animated for all of their possible actions using 3D Studio Max's key – framing animation techniques.

The idle model is first loaded. It is possible to animate the position, rotation, and scale of a model, and almost any parameter setting that affects the object's shape and surface. It is possible in link objects for hierarchical animation, using both forward and reverse kinematics, and to edit the animation in Track View.

6.2 ANIMATION

6.2.1 KEY FRAMES

Key Frames are the ones, where one can define the animation for a parameter by specifying its exact value at a given set of times. The software can then work out by interpolating what the value should be between the keys.

6.2.2 TIME CONTROLS

Time Controls can be found on the lower interface bar between the key controls and the Viewport Navigation Controls. The Time Control

buttons include buttons to jump to the Start or End of the animation, or to step forward or back by a single frame. You can also jump to an exact frame by entering the frame number in the frame number field. The Time Controls also include the Time Slider found directly under the view ports.

6.2.3 TIME SLIDER

The Time Slider provides an easy way to move through the frames of an animation. To do this, just drag the Time Slider button in either direction. The Time Slider button is labelled with the current frame number and the total number of frames. The arrow buttons on either side of this button work the same as the Previous and Next Frame (Key) buttons.

6.2.4 TRACK BAR

The Track Bar is situated directly under the Time Slider. The Track Bar displays a rectangular marker for every key for the selected object. These markers are colour-coded depending on the type of key. Position keys are red, rotation keys are green, scale keys are blue, and parameter keys are dark grey. The current frame is also shown in the Track Bar as a light blue transparent rectangle.

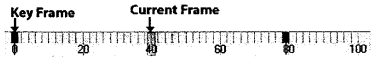

Fig 6.1: Track Bar

The Track Bar shows key markers only for the currently selected object or objects, and each marker can represent several different keys. When the mouse is moved over the top of these markers, the cursor changes to a plus sign and a marker can be selected by clicking on it. Selected markers turn white. Using the Ctrl key, one can select multiple

keys at the same time. Selection of multiple key markers can be done by clicking an area of the Track Bar that contains no keys and then dragging an outline over all the keys to be selected. Moving the cursor over the top of a selected key, the cursor is displayed as a set of arrows enabling to drag the selected key to the left or right. While dragging a key hold the 'Shift' button to create a copy of the key. Pressing the Delete key deletes the selected key.

6.2.5 ANIMATION WITH KEY FRAMES

Keys define a particular state of an object at a particular time. Animations are created as the object moves or changes between two different key states. The easiest way to make keys is using the Key Controls. These controls are located to the left of the Time Controls.

Max includes two animation modes: **Auto Key** and **Set Key**. One can select either of these modes by clicking the respective buttons at the bottom of the interface. When active, the button turns bright red, and the border around the active viewport also turns red to remind that it is in animate mode. Red also appears around a spinner for any animated parameters whose value changes as the track bar slides.

6.2.6 THE AUTO KEY MODE

With the Auto Key button is enabled, every transformation or parameter change creates a key that defines where and how an object should look at that specific frame.

To create a key, drag the Time Slider to a frame where a key is needed and then animate by moving or scaling or rotating the selected

6.3 SYNCHRONISATION

When a character in the game is taken, an animation for its idle, walk, attack, impact, die states are animated and saved as separate 3D Studio Max Mesh (.3ds) files. These files are converted to DirectX Model (.x) file format. Now these created character animations are to be synchronised in the game so as to give the illusion of the character walking, attacking, etc. The idle state is the default state of almost all models. That state is mostly active and displayed on screen. This state is switched to walk state or attack state, a temporary switching, to which the character transforms for a short time due to an appropriate input by the player, like the cursor keys for walk.

6.3.1 PRINCIPLE APPLIED

The principle used in games is the same as with televisions. Human eye can retain the last seen frame for a few micro seconds more, even after it's gone out of the eye's focus. The frame persists for a bit longer during which the next new frame is displayed. Thus the required frames are cycled so as to give the illusion that something really happens, believable to human eyes.

First the idle state animation of a character is loaded and displayed. Then the other states' animations like walk, attack, etc. are appended to the character, but hidden from view by cycling the frames only till the move state's animation, thus not displaying the remaining appended states beyond it. When the player instructs the character to be moved to a new location, then the move state animation frames are cycled, along with the whole object is constantly moved to the target location, thus giving the illusion that the character really walks on screen.

object or change a parameter value associated with it, and a key is automatically created. When the first key is created, Max automatically goes back and creates a key for frame 0 that holds the object's original position or parameter. Upon setting the key, Max then interpolates all the positions and changes between the keys. The keys are displayed in the Track Bar.

Each frame can hold several different keys, but only one for each type of transform and each parameter. For example, move, rotate, scale, and change the Radius parameter for a sphere object with the Auto Key mode enabled, then separate keys are created for position, rotation, scaling, and a parameter change.

6.2.7 THE SET KEY MODE

The Set Key button offers more control over key creation and sets keys only when Set Key button is clicked. It also creates keys only for the key types enabled in the Key Filters dialog box. The Key Filters dialog box can be opened, by clicking the Key Filters button. Available key types include All, Position, Rotation, Scale, IK Parameters, Object Parameters, Custom Attributes, Modifiers, Materials, and Other (which allows keys to be set for manipulator values).

6.2.7 SETTING KEY FRAME

Once the model or any other primitive object for the animation is ready, press the "Auto Key" button in the bottom right-hand corner of the screen. The "Auto Key" button will turn red. While it's red, anything you change will create a key frame in that state at that frame.

A slider at the bottom of the screen tells the current frame. It should be at 0. It is dragged to the frame where the sphere needs to be in its new position by. Say 25 frames, for instance. Now the sphere is moved or rotated or scaled, with respect to the animation needed.

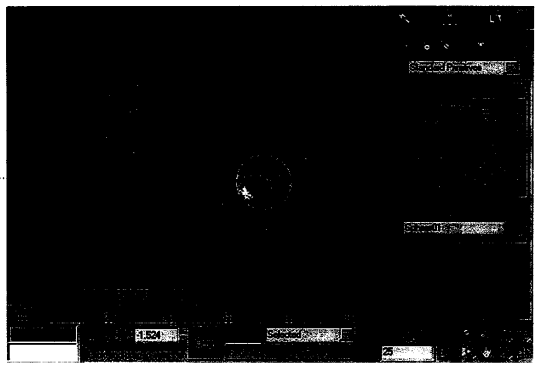

Fig 6.2: Animation in 3ds Max

The animation is done now. The play button is clicked to watch the animated sphere in action. When the track bar is moved to a new frame and the sphere moved, then a third new key frame will be created with the sphere at its new position. Turn the animate mode off and move the sphere it moves the entire animation by that displacement. When the play button is clicked, the whole animation is looped in the viewport.

Thus the required model is animated.

6.3 SYNCHRONISATION

When a character in the game is taken, an animation for its idle, walk, attack, impact, die states are animated and saved as separate 3D Studio Max Mesh (.3ds) files. These files are converted to DirectX Model (.x) file format. Now these created character animations are to be synchronised in the game so as to give the illusion of the character walking, attacking, etc. The idle state is the default state of almost all models. That state is mostly active and displayed on screen. This state is switched to walk state or attack state, a temporary switching, to which the character transforms for a short time due to an appropriate input by the player, like the cursor keys for walk.

6.3.1 PRINCIPLE APPLIED

The principle used in games is the same as with televisions. Human eye can retain the last seen frame for a few micro seconds more, even after it's gone out of the eye's focus. The frame persists for a bit longer during which the next new frame is displayed. Thus the required frames are cycled so as to give the illusion that something really happens, believable to human eyes.

First the idle state animation of a character is loaded and displayed. Then the other states' animations like walk, attack, etc. are appended to the character, but hidden from view by cycling the frames only till the move state's animation, thus not displaying the remaining appended states beyond it. When the player instructs the character to be moved to a new location, then the move state animation frames are cycled, along with the whole object is constantly moved to the target location, thus giving the illusion that the character really walks on screen.

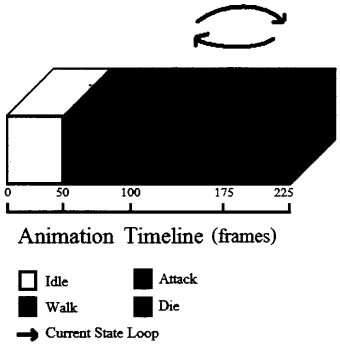

Fig 6.3: Animation – Synchronisation Principle

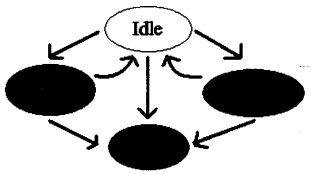

Fig 6.4: State Chart of a Character

6.3.2 LOADING AND APPENDING OF STATES

Load object call is made for the first time to assign an Object ID. After an object is created with that ID, then further states are appended to it by a call to Append object function.

6.3.3 SETTING ANIMATION SPEED

An object animation that is slower in 3DS Max may run faster when played using DarkBasic's DirectX call or vice-versa, this is because of the faster/slower frame rate of a game, than of the default movie animation of Studio Max. So it's always better to set the animation's speed before playing it.

6.3.4 PLAYING/LOOPING THE ANIMATION

Once the animation speed is set the animation can be played once or looped as a repeated cycle, until halted using stop animation. Animation looping is required in many points, almost all, of the game since the basic movement itself is based on the animation that is looped until the movement ceases.

6.3.5 A CHARACTER'S LIFE CYCLE

A character in the game comprises of a model associated to it which is loaded with the respective animations associated to it. These animations have all action sequences doable by the character. Once loaded and animations are appended the speed of animation is set.

Then in the main loop of the game, which runs repeatedly until the game is quit, input for a move command on the character is scanned. If a new location is set for that character then the idle state animation is hided and the walk animation is looped, simultaneously moving its model in the real world plane. Once the required target position is reached the walk sequence of the model is hided and the idle state sequence is loaded again.

In the main loop of the game, every character is checked for its collision with other characters. If a collision between an enemy model and hero's bullet occurs, then the life variable of that character is reduced according to arm used by the player.

If the health variable of the character reaches 0 then the character's death sequence is played on its model and is unloaded from the memory.

GAME – CONTROLS AND OBJECTIVES

CHAPTER 7

GAME – CONTROLS AND OBJECTIVES

7.1 INPUT CONTROLS

Operations like collection of health, cartridges, are automated i.e. the collection is automatically credited once the hero object touches or walks over them.

Action	Input
Positioning Movement	
Move Forward	Cursor Up (?)
Move Back	Cursor Down (?)
Move Left	Cursor Left (?)
Move Right	Cursor Right (?)
Angular Movement	
Look Up	Mouse Steer Up
Look Down	Mouse Steer Down
Turn Left	Mouse Steer Left
Turn Right	Mouse Steer Right
Combat Controls	
Attack	Left Mouse Click
Reload Weapon	Right Mouse Click
Weapon Swap Controls	
Load Pistol	1
Load Short Gun	2
Load Machine Gun	3
Load Sniper Rifle	4
Sniper Rifle Controls	
Zoom In	Z
Zoom Out	Enter

Table 7.1: Game Input Controls

7.1.1 POSITIONING & ANGULAR CONTROLS

These are the general hero model manipulating commands such as move forward, backward, etc for hero's locomotion. Angular movements like look up, look down, etc are for directional changes and looks.

7.1.2 COMBAT CONTROLS

These are for attack & reload operations, with respect to the current weapon selected. The attack operation won't occur if there aren't enough bullets for the current weapon. Likewise, the reload operation won't occur if there aren't enough cartridges for the selected weapon.

7.1.3 SNIPER RIFLE CONTROLS

These are Sniper Rifle specific commands i.e. these will work only if the Sniper Rifle is selected. Z is to focus the target using the zoom operation that the rifle provides. The view on screen shrinks to a small viewfinder of the rifle. Enter is to return back to normal mode.

7.2 GAME PLAY & OBJECTIVES

The game play is of first person shooter (FPS) interface i.e. 3D view of the virtual world for the player whose view will be a simulated view of the hero object. The game play has a night environment with a dim ambient light thought out the world. The game play is by the input that can be fed using a keyboard and a mouse. The output can be visualized using any 1024 x 768 resolution supporting monitor. The audio and special effects can be heard using any speakers that can be jacked-in to the sound card.

7.2.1 PRIMARY OBJECTIVES

- The hero should be alive through out the game
- To clear a level all enemies are to be neutralised
- To move to the next level a key should be obtained as clearance for that level, which will appear at a random location, once all enemies are neutralised.

7.2.2 SECONDARY OBJECTIVES

- Obtain all ammunition for survival
- For full health restoration collect the heart objects in all levels
- Use sniper rifle for targets that are at a long distance and difficult to eliminate

7.3 GAME STATUS INDICATORS

The top left bar (Health Bar) indicates the player health which when reaches 0 triggers game over. The currently encountered enemy's life is indicated by the bar (Enemy Health) next to the health bar. Above this bar is the count that shows the number of enemies eliminated / total number of enemies.

The current arm selected is displayed in the top right corner with the number of bullets in it. The number of cartridges for that weapon is also displayed below it.

CONCLUSION

CHAPTER 8

CONCLUSION

This game will be a whole new experience for the young gamers who are constantly in search for something newer and better with breathtaking surrealism. This game has been tested on various systems for its performance in different architectures.

Feedbacks where collected from a few gamers (friends) who actually played it throughout the end of the game, and was found to be good, with certain tips for enhancements.

Though 2D, Role Playing Games and animations haven't lost its charm yet, it can be foreseen that the upcoming era belongs to the 3D perspective, which is much wider, clearer and extensible. Keeping this in mind the project opens up new avenues for a whole new experience in the 3D gaming arena.

POSSIBLE FUTURE ENHANCEMENTS

CHAPTER 9

POSSIBLE FUTURE ENHANCEMENTS

The first and foremost enhancement possible for the game can be the refinement of the overall engine to make it faster and better. This can be done by less complex objects and reduction of the number of objects loaded for a single stage of the game. It gives a richer look and much lesser response time that keeps the player glued to the game.

Network play (Multiplayer) functionality can be a thoroughly amazing enhancement for the game, which enable more than one player to play simultaneously either as teams or against themselves. Chat facility between the players can also be provided for better communication between players to device game plans.

An additional view other than the default 3D view can be provided for the player e.g. 3rd person shooter. This allows the player to have a better view of the arena they are in. Rotation and titling of the whole view point can also be added to the engine which adds more reality to the gaming experience.

APPENDICES

APPENDIX 1

3DS FILE FORMAT ARCHITECTURE (.3DS)

The 3ds file format is made up of *chunks*. Chunks describe what information is to follow and what is made up of; its ID and the location of the next block. The next chunk pointer is relative to the start of the current chunk and in bytes. The binary information in the 3ds file is written in a special kind of way. Namely the least significant byte comes first in an int. For example: 4A 5C 5C 3B 8F where 5C 4A is the low word and 8F 3B is the high word.

Table a1: Format of a Chunk

Start	End	Size	Name
0	1	2	Chunk ID
2	5	4	Pointer to next chunk relative to the place where Chunk ID is, in other words the length of the chunk

Listing a2: Sample 3ds file's hierarchy

```
MAIN3DS   (0x4D4D)
|
+--EDIT3DS   (0x3D3D)
|  |
|  +--EDIT_MATERIAL (0xAFFF)
|  |  |
|  |  +--MAT_NAME01 (0xA000)
|  |
|  +--EDIT_CONFIG1   (0x0100)
|  +--EDIT_CONFIG2   (0x3E3D)
|  +--EDIT_VIEW_P1   (0x7012)
|  |  |
|  |  +--TOP              (0x0001)
|  |  +--BOTTOM           (0x0002)
|  |  +--LEFT             (0x0003)
|  |  +--RIGHT            (0x0004)
|  |  +--FRONT            (0x0005)
|  |  +--BACK             (0x0006)
|  |  +--USER             (0x0007)
|  |  +--CAMERA           (0xFFFF)
|  |  +--LIGHT            (0x0009)
|  |  +--DISABLED         (0x0010)
|  |  +--BOGUS            (0x0011)
```

Chunks have a hierarchy imposed on them that is identified byte it's ID. A 3ds file has the Primary chunk ID 4D4Dh. This is always the first chunk of the file. Within the primary chunk are the main chunks.

A preview and a reference to the hierarchy of chunks, above listing (a) is to show the different chunk ID's and their place in the file. The chunks are given a name because below the diagram is a list which defines the names to the actual chunk id's. This makes it easier to put it in some source code (how convenient that some sample code is included)

APPENDIX 2

DIRECTX FILE FORMAT ARCHITECTURE (.X)

The DirectX file format is an architecture- and context-free file format. It is template-driven and is free of any usage knowledge. The file format may be used by any client application and currently is used by Direct3D Retained Mode to describe geometry data, frame hierarchies, and animations.

This following are the sections of the file format. The file format uses the extension .x when used with the DirectX Software Development Kit (SDK) or other programming file references.

- Reserved Words
- Header
- Comments
- Templates
- Data
- Use of Commas and Semicolons

A SIMPLE CUBE IN .X FILE FORMAT

This file defines a simple cube that has four red sides and two green sides. Notice in this file that optional information is being used to add information to the data object defined by the Mesh template.

```
Material RedMaterial {
1.000000;0.000000;0.000000;1.000000;;    // R = 1.0, G = 0.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
}
Material GreenMaterial {
0.000000;1.000000;0.000000;1.000000;;    // R = 0.0, G = 1.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
}
// Define a mesh with 8 vertices and 12 faces (triangles). Use
// optional data objects in the mesh to specify materials, normals,
// and texture coordinates.
Mesh CubeMesh {
8;                              // 8 vertices
1.000000;1.000000;-1.000000;,   // vertex 0
-1.000000;1.000000;-1.000000;,  // vertex 1
-1.000000;1.000000;1.000000;,   // etc...
1.000000;1.000000;1.000000;,
1.000000;-1.000000;-1.000000;,
-1.000000;-1.000000;-1.000000;,
-1.000000;-1.000000;1.000000;,
1.000000;-1.000000;1.000000;;

12;                             // 12 faces
3;0,1,2;,                       // face 0 has 3 vertices
3;0,2,3;,                       // etc...
3;0,4,5;,
3;0,5,1;,
3;1,5,6;,
3;1,6,2;,
3;2,6,7;,
3;2,7,3;,
3;3,7,4;,
3;3,4,0;,
3;4,7,6;,
3;4,6,5;;

// All required data has been defined. Now define optional data
// using the hierarchical nature of the file format.
MeshMaterialList {
2;                              // Number of materials used
12;                             // A material for each face
0,                              // face 0 uses the first
```



```
0,                              // material
0,
0,
0,
0,
0,
0,
1,                              // face 8 uses the second
1,                              // material
1,
1;;
{RedMaterial}                   // References to the definitions
{GreenMaterial}                 // of material 0 and 1
}
MeshNormals {
8;                              // define 8 normals
0.333333;0.666667;-0.666667;,
-0.816497;0.408248;-0.408248;,
-0.333333;0.666667;0.666667;,
0.816497;0.408248;0.408248;,
0.666667;-0.666667;-0.333333;,
-0.408248;-0.408248;-0.816497;,
-0.666667;-0.666667;0.333333;,
0.408248;-0.408248;0.816497;;
12;                             // For the 12 faces,
3;0,1,2;,                       // define the normals
3;0,2,3;,
3;0,4,5;,
3;0,5,1;,
3;1,5,6;,
3;1,6,2;,
3;2,6,7;,
3;2,7,3;,
3;3,7,4;,
3;3,4,0;,
3;4,7,6;,
3;4,6,5;;
}
MeshTextureCoords {
8;                              // Define texture coords
0.000000;1.000000;              // for each of the vertices
1.000000;1.000000;
0.000000;1.000000;
1.000000;1.000000;
0.000000;0.000000;
1.000000;0.000000;
0.000000;0.000000;
1.000000;0.000000;;
}
}
```

APPENDIX 3

SAMPLE CODING

```
Sync On
Sync Rate 30
hide mouse

camvalue=200
Backdrop on
Set camera range 1,5000

Fog on
Fog distance 3000
fog color RGB(25, 25, 25)
Color Backdrop RGB(0, 0, 0)

set ambient light 75

Rem make matrix
matrix_x=20000
matrix_y=30000
Make matrix 1,matrix_x,matrix_y, 50, 75

Rem texture matrix
Load image "Model\Ground.bmp",2
Prepare matrix texture 1, 2, 2, 2

currentArm=9000
swapArm(currentArm, 0)

Rem intialize particle counter
Pn1=20

dim LifeRecord(9,12)
Life=0
for i=0 to 12
   for j=0 to 9
   LifeRecord(j,i)=Life
   next j
next i

make object box 7, 60,390,40
set object collision on 7
hide object 7

`setting up sniper viewfinder
set image colorkey 255, 255, 255
load image "Model\viewfinder.bmp", 701
set sprite 2,0,1
sprite 2, 0, 0, 701
hide sprite 2

`setting up crosshair
set image colorkey 255,0,255
load image "Model\crosshair.bmp", 700
`rem Setup crosshair sprite
set sprite 1,0,1
```



```
sprite 1,screen width()/2-15,screen height()/2,700

load music "Effects\CRICKETS.WAV",6000

Rem Make bullet
Make Object Sphere 2, .05
Hide Object 2
automatic object collision 2, .05, 0

Make Object Sphere 3, 20
set object collision on 3
Load Image "Model\bgalla1.jpg",22
Texture object 3,22


Rem load particles
For x = 0 to 10
   Make object plain x+10,5,5
   Texture object x+10,22
   Set object x+10,1,0,0
   Ghost object on x+10
   set object collision off x+10
Next x

Rem load sound effect
load sound "Effects\Gun.wav", 22

Rem start point of player object
X#=5700
Z#=1400
Y# = Get ground height(1,X#,Z#)

position object 7, X#, 0, Z#
Position Camera X#,Y#+camvalue, Z#
bulletVelocity=60

loop music 6000

Rem load gun picture
load image "Effects\Pistol.png", 40

set sprite 3, 0, 1
sprite 3, 500, -10, 40

Rem Main loop
Do
   OldCamAngleY# = CameraAngleY#
   OldCamAngleX# = CameraAngleX#

   CameraAngleY# = WrapValue(CameraAngleY#+MousemoveX()*0.2)
   CameraAngleX# = WrapValue(CameraAngleX#+MousemoveY()*0.2)
   CameraAngleZ# = Camera angle Z()

   colli=object collision(7,0)

   if colli=0
      lastUnColliX#=X#
      lastUnColliZ#=Z#
        Rem  Control input for camera
      If Upkey()=1
       XTest# = Newxvalue(X#,CameraAngleY#,50)
```



```
       ZTest# = Newzvalue(Z#,CameraAngleY#,50)
         If XTest#>0 and XTest#<matrix_x and ZTest#>0 and
ZTest#<matrix_y
            X#=XTest#
            Z#=ZTest#
          Endif
      Endif
      If Downkey()=1
         XTest# = Newxvalue(X#,Wrapvalue(CameraAngleY#-180),10)
         ZTest# = Newzvalue(Z#,Wrapvalue(CameraAngleY#-180),10)
         If XTest#>0 and XTest#<matrix_x and ZTest#>0 and
ZTest#<matrix_y
            X#=XTest#
            Z#=ZTest#
         Endif
      Endif

      If Leftkey()=1
       XTest# = Newxvalue(X#,Wrapvalue(CameraAngleY#-90),10)
       ZTest# = Newzvalue(Z#,Wrapvalue(CameraAngleY#-90),10)
       If XTest#>0 and XTest#<matrix_x and ZTest#>0 and
ZTest#<matrix_y
         X#=XTest#
         Z#=ZTest#
       Endif
      Endif

      If Rightkey()=1
       XTest# = Newxvalue(X#,Wrapvalue(CameraAngleY#+90),10)
       ZTest# = Newzvalue(Z#,Wrapvalue(CameraAngleY#+90),10)
       If XTest#>0 and XTest#<matrix_x and ZTest#>0 and
ZTest#<matrix_y
         X#=XTest#
         Z#=ZTest#
       Endif
      Endif

      if delay<>0 then delay=delay-1

      if Mouseclick()=1 and BulletLife=0 and delay=0
         if currentArm=9000
            if pistol>=1
               pistol=pistol-1
               BulObj=pistol
               delay=25
            else
               goto shoot_quit:
            endif
            play sound 22
         endif
         if currentArm=9001
            if short>=1
            short=short-1
            delay=50
            BulObj=short
            else
            goto shoot_quit:
            endif
            play sound 23
         endif
         if currentArm=9002
```



```
            if mgun>=1
               mgun=mgun-5
               delay=5
               BulObj=mgun
            else
               goto shoot_quit:
            endif
         play sound 24
         endif
         if currentArm=9003
            if sniper>=1
               sniper=sniper-1
               delay=75
               BulObj=sniper
            else
               goto shoot_quit:
            endif
         endif
         if sniperMode=1
            Position object 2,RealposX#+(X#-
ForwardX#),Y#+190,RealposZ#+(Z#-ForwardZ#)
         else
            Position object 2,X#,Y#+190,Z#
         endif

         Set object to camera orientation 2
         BulletLife=25
         set object speed currentArm, 250
         play object currentArm, 51, 75

      Endif
   shoot_quit:

   if currentArm=9003 and keystate(44) and sniperMode=0
      hide object 9003
      hide sprite 1
      show sprite 2
      RealposX#=X#
      RealposZ#=Z#
      XTest# = Newxvalue(X#,CameraAngleY#,1500)
      ZTest# = Newzvalue(Z#,CameraAngleY#,1500)
      If XTest#>0 and XTest#<matrix_x and ZTest#>0 and ZTest#<matrix_y
         X#=XTest#
         Z#=ZTest#
      Endif
      ForwardX#=X#
      ForwardZ#=Z#
      sniperMode=1
   endif

   if (returnkey() and sniperMode=1)
      hide sprite 2
      show object 9003
      show sprite 1
      X#=RealposX#+(X#-ForwardX#)
      Z#=RealposZ#+(Z#-ForwardZ#)
      sniperMode=0
   endif

else
    X#=lastUnColliX#
```

```
    Z#=lastUnColliZ#
endif


Rem Rotate camera
cTestX#=WrapValue(CameraAngleX#-180)
if cTestX# > 225 then CameraAngleX#=45
if cTestX# < 135 then CameraAngleX#=315

Yrotate camera CurveAngle(CameraAngleY#,OldCamAngleY#,24)
Xrotate camera CurveAngle(CameraAngleX#,OldCamAngleX#,24)

   if mouseclick()=2 and reLoad=0
      if currentArm=9000 and pistol_cart>=1
         pistol=10
         pistol_cart=pistol_cart-1
         BulObj=pistol
         delay=0
         set object speed currentArm, 80
         play object currentArm, 76, 100
         reLoad=1
      endif
   endif


   If BulletLife > 0

      Dec BulletLife
      Move object 2, bulletVelocity

      `likewise for all weapons
      if currentArm=9001
         For x = 21 to 30
            show object x
         Next x
         obX#=Object position X(2)
         obY#=Object position Y(2)
         obZ#=Object position Z(2)
      endif

      ObjectLife(pwd)
      if object collision(2, 0) > 0
         BulletLife=0
         position object 2, -100,-100,-100
         For x = 21 to 30
            hide object x
         Next x
      endif

      if BulletLife = 0
         position object 2, -100,-100,-100
         if currentArm=9001
            For x = 21 to 30
               hide object x
            Next x
         endif
      endif
   Endif
   Sync
Loop
```


APPENDIX 4

SCREENSHOTS