Project Final Report (2002 – 2003)
Virtual Reality over the Internet
Virtual Reality Fighter
CSIS Group

```
Received <Player_id_Request>
Generating new player ID
Received <Session_Start_Request>
Starting Session
Player 0 changed to (0, 0, 0)
Player 1 changed to (0, 1, 2.34)
Player 0 fired
Player 1 not shot
```

```
Starting rendering
Texture prepared
Display list built
Lighting enabled
Material enabled
Speed: 100frm/s
Terrain fractal ready
Model constructed
VRML parsed
XML parsed
```

Lee Hoi Sing, James
Leung Shun Kit, Frank
Ma Chi Ho, Denny
Ng Sze Lung, Raymond
Suo Jingji, Alex

CSIS Final Year Project (2002 – 2003)
Virtual Reality over the Internet

Virtual Reality Fighter

Final Report

Lee Hoi Sing, James
Leung Shun Kit, Frank
Ma Chi Ho, Denny
Ng Sze Lung, Raymond
Suo Jingji, Alex

Contents

Chapter 1 Introduction

Have you thought of playing an interactive, real-time online game with hundreds of other players together? Virtual Reality Fighter, an Internet game applying virtual reality techniques, allows users to interact with other players in real-time. Players in the game will control their own fighters and join the fantastic game world at any place and any time. The game is able to support hundreds of players concurrently and to deliver high quality computer graphics.

1.1 Objectives of the Project

The objective of the project is to build a ***scalable***, ***real-time***, ***flexible 3-D online game*** with ***good graphic quality***. This project stresses on several key areas in the *Computer Graphic*, *Internet Communication* and *Object Definition*. In the field of Computer Graphic, the focus includes collision detection, terrain generation, performance tuning, and so on; in the field of Internet Communication, the focus includes Multi-Server Architecture, Synchronization between clients and servers and Partition of clients. In the field of Object Definition, the focus is VRML and XML.

1.2 Fundamental Requirements

1. Real-time interaction between clients.
2. Real-time graphics with at least 25 fps.
3. Support large amount of users (hundreds users).
4. Network traffic is minimized and normal broad band users can play this game at least as clients.
5. Models can be dynamically added to the game (optional in program implementation).

1.3 Overview of the game

Players can select different fighter models and terrains before the game started, and they can join any game server on the Internet. After joining the server, players will control their own fighter, and fly freely in a 3D virtual scene. Their ultimate objective is to destroy all opponents using laser and missiles.

Chapter 2 Overall System Architecture

The game system is applying the client-server architecture. The system is divided into two tiers: the client side and the server side. The basic organization of the clients and servers is shown in the figure below:

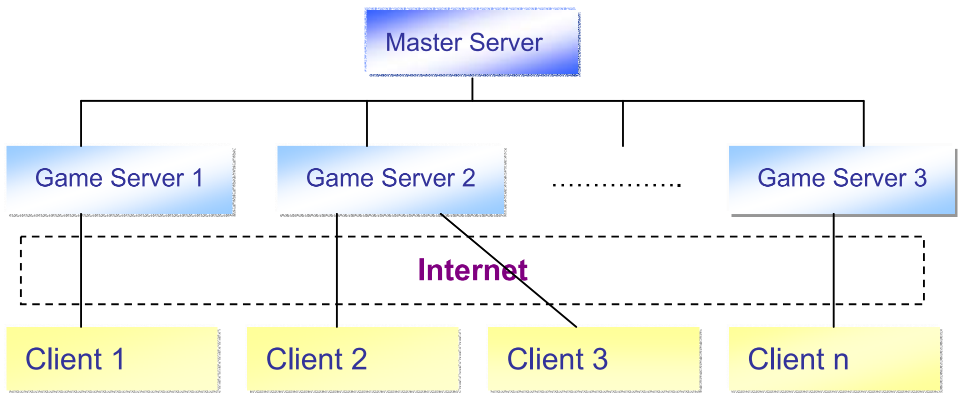


Fig. 2.1 multiple server architecture

2.1 Server

At the server side, there is one master server which controls and maintains the global game information such as the number of game servers, number of clients, etc. The master server is responsible for the co-ordination between game servers and the initial negotiation with the clients when they join the game.

For the individual game servers, they are responsible for maintaining the state and determining the game logic for their individual partitions. Each server will keep the information of a particular partition. Therefore, the entire game state is distributed among the game servers.

2.2 Client

At the client side, the client communicates with one particular server at any time, depends on which partition the client is in. The client is also responsible for rendering the 3D game environment by utilizing the game state information received from the server and the static fighter and terrain models in the client side. Further details for the game client and server will be given in the following sections.

Chapter 3 Game Engine

3.1 Overview

The game engine is the core of the system. It is responsible for coordinating different modules within the system and controlling the flow of the whole game. Since the overall system is divided into client and server, the game engine for client and server are different. In the following section, the game engines for the client and server will be presented individually.

3.2 Architecture

3.2.1 Server

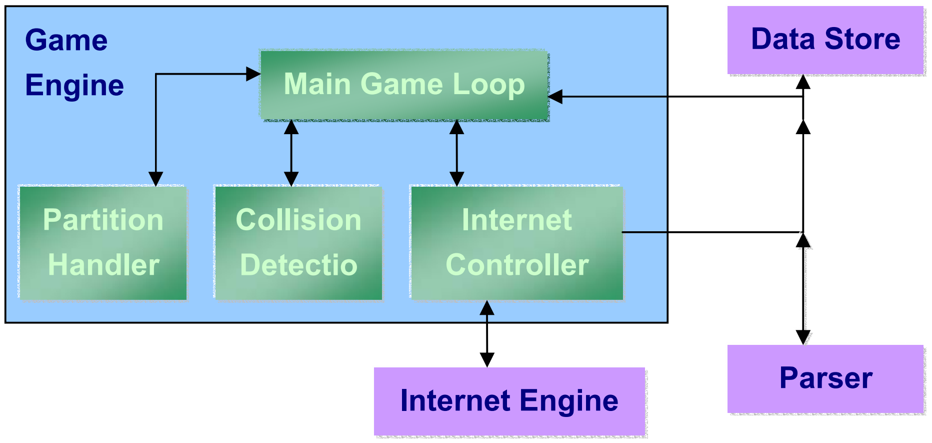


Fig. 3.1 server architecture

The Game Engine of the server includes the Main Game Loop, the Internet Controller, the Partition Handler and the Collision Detection Module. Also, the game engine will collaborate with the Internet Engine and Parser, and manipulate the data in the Data Store, to achieve the functionalities that need to be provided by the server.

The basic functionality of each component in the Server Game Engine is:

Main Game Loop: Control the flow of the game and coordinate the game rules. This loop will process all the game messages and determine appropriate actions based on the pre-defined game rules. The actions will be carried out by utilizing the components within the Game Engine and/or modules cooperate with Game Engine.

Internet Controller: Co-ordinate the internet message communication between game servers as well as between servers and clients. In the server side, the Internet Controller contains a synchronizer ensuring that all the messages exchanged between clients and servers are in proper order. More detail description on the Synchronizer will be given in section 4.4.

Collision Detection Module: Determine whether two objects in the game world collide or not. The collision detection module will detect the fighter-fighter collision, fighter-laser collision, fighter-missile collision and fighter-terrain collision. More detail description on the Collision Detection Module will be given in section 5.5.

Partition Handler: Determine which partition the client belongs to and maintain all the partition information to reduce the network traffic. More detail description on the Partition Handler will be given in section 4.5.

Data Store: Game database storing all the static 3D-fighter models and dynamic game entity information. The game state is stored in the Data Store and will be manipulated by the Game Engine.

Parser: Responsible for parsing the 3D-models in VRML format and the self-defined XML fighter object into the game system.

Internet Engine: Responsible for handling the internet connection between different game servers and between the clients and server. It is also responsible for transferring the game messages between clients and servers.

3.2.2 Client

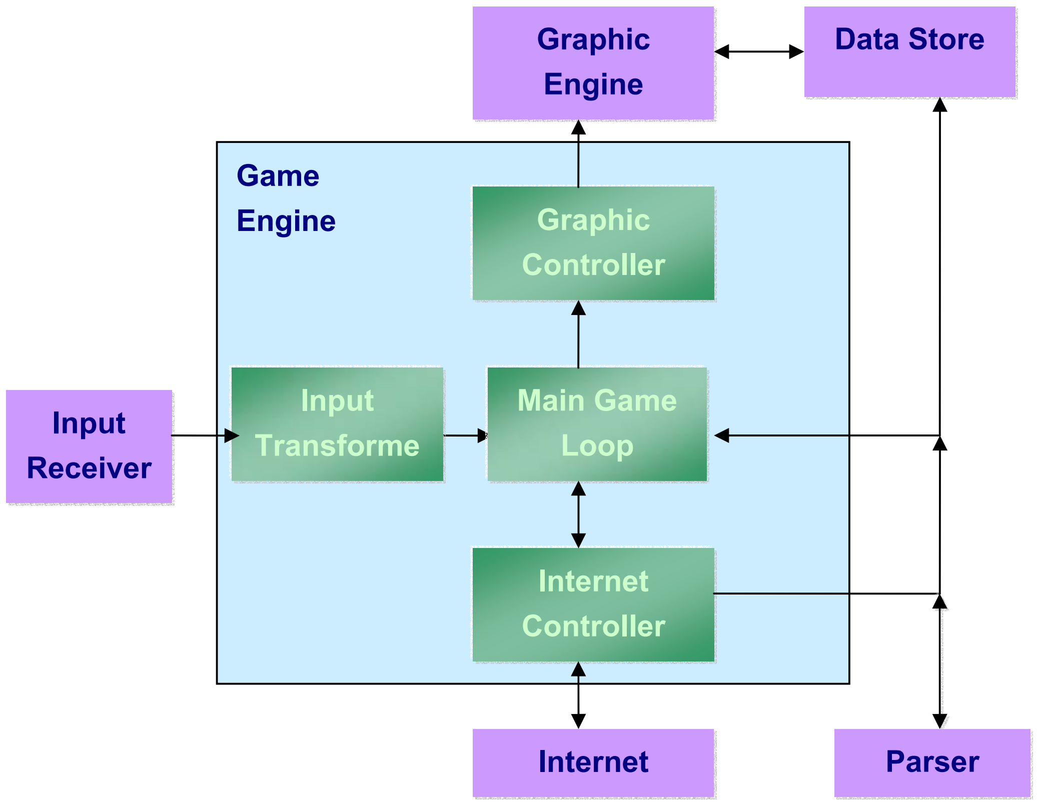


Fig 3.2 Client architecture

The Game Engine of the client is more complicate than the server. It includes the Main Game Loop, Internet Controller, Graphic Controller and Input Transformer. The other modules collaborating with the Game Engine are the Input Receiver, Internet Engine, Graphic Engine, Parser and the Data Store.

The basic functionality of each component in the Client Game Engine is:

Main Game Loop: Control the flow of the game, render the virtual 3D environment and handle the user input. This loop will process all the user commands. Some game messages will be processed locally and some will be sent to the server. Also, the game loop will process the game messages received from the server and update the game state.

Graphic Controller: Responsible for coordinating the rending of 3D-game environment. It acts as a bridge between the game logic and the Graphic Engine.

Internet Controller: Co-ordinate the internet message communication between game servers and clients.

Input Transformer: Responsible for transforming the user inputs into the internal game messages that can be processed by the Game Engine.

Graphic Engine: Render the 3D virtual game environment. It will render the game environment based on the game information stored in the Data Store. More detail description on the Graphic Engine will be given in section 5.

Data Store: Game database storing all the static 3D-fighter models and dynamic game entity information. The game state is stored in the Data Store and will be manipulated by the Game Engine.

Parser: Responsible for parsing the 3D-models in VRML format and the self-defined parameters in XML for fighter, missile and terrain objects into the game system.

Internet Engine: Responsible for handling the internet connection between different game servers and between the clients and server. It will also responsible for transferring of game messages between clients and servers.

Input Receiver: Responsible for capturing the user inputs from the keyboard and mouse. All the inputs will be kept in an internal buffer for later processing.

3.3 Game Flow

3.3.1 Server

The flow of the game at the server side can be described by the following diagram.

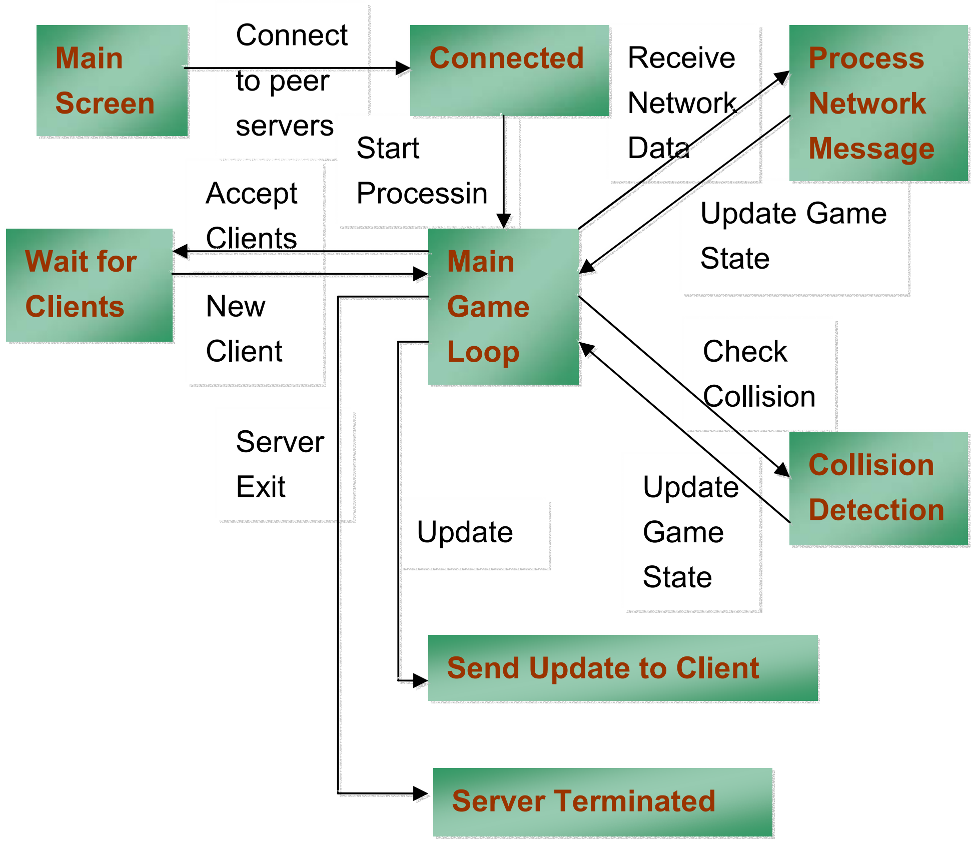


Fig 3.3 Game flow of server

At the server side, after the server is started, the clients can connect to the master server, which will provide an array of peer game servers that the current game server needed to connect. Also, the master server will determine the partition which the current server needed to handle.

After the current game server had connected to all the peer game servers, it can start the processing. There are several tasks the server needed to handle, they are listed below:

Wait for Client

The server needed to wait for the connection from the clients. After a client is connected, the server needs to prepare all the necessary information required to allow the client to join the game. For instance, the server needs to generate the player ID, the initial position of the player, etc. Also, the server needs to inform the client about the other players within the same partition.

Process Network Message

Clients will periodically send the game messages to the server and the server needs to process the game messages and update the game state. Based on different message types, the server may be required to perform other tasks such as collision detection.

Collision Detection

Collision will happen when the client fighters collide with the terrain, the other clients, the missile and the laser. These collisions are detected by the server and the server will perform the detection based on the position information of the clients received from the network and/or the message which indicate a missile or laser is fired.

Send Update to Clients

The server needed to send the updated game state information to all the clients so that the client can render the correct scene in real time. The update is send to clients periodically.

When a server is terminated, it needs to inform the master server it will terminate and all the clients that the server is handling currently will need to be transferred to other available servers.

There is a special case that the peer game servers needed to communicate. The peer servers will only communicate when there is a client went from one partition to another. Then the original game server needs to inform the other server that a client is going to change partition and the new server needs to prepare for this.

3.3.2 Client

The flow of the game at the client side can be described by the following diagram:

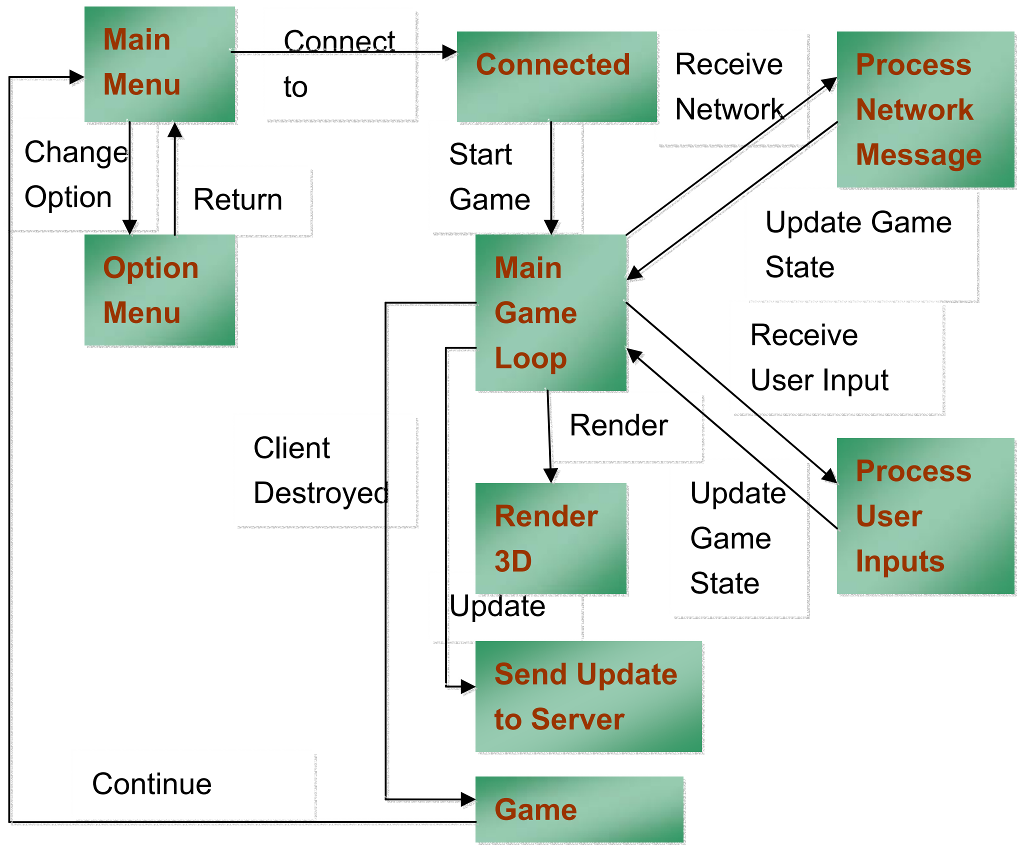


Fig. 3.4 Game flow of client

On the client side, when the program is started, a main menu will be shown on the screen. The player can change the game options on the option menu. The player can choose different fighter models in the option menu and change the key mapping that is used to control the fighter.

Before the player can start the game, he/she needed to connect to one of the game server on the Internet. The following shows the procedures to connect to the game server.

1. The client connects to the master server on the Internet.
2. The master will generate a player ID for the client and a list of game servers that are available on the Internet
3. The client performs internet connection delay test to find out which server on the server-list with the minimum delay.
4. The client will connect to the server with minimum delay.

After the client is connected to one of the server, the player can start the game. The client will perform the following tasks during the game:

Process Network Message
Server will periodically send the most update game state to the clients. The client needs to process all those messages and update its game state.

Process User Input
The player will input commands through the keyboard or mouse in response to the change in the game environment. The game loop needs to handle all the user input and reflect those inputs into the game world.

Render 3D scene
The game environment change in real time and the game loop needs to render the updated 3D environment periodically. Otherwise, the player may give incorrect response due to the delay on the rendering of the environment.

Send Update to Server
The game state on the client side will change according to the player’s input. Those changes in the game state need to be reflected in the server’s game state as well as other client’s game state. That means all the game state on the server and client should be synchronized. Therefore, the client will send the update to the server frequently.

During the game, the player can control his/her fighter and try to destroy as many opponents as he/she can. The game will end when the player’s fighter is destroyed by the other player. Then the game will return to the main menu and the player can start over again.

3.4 Features

3.4.1 Real-Time Interaction

The design of the game engine and game flow allow the real-time interaction between clients. All the change in the game world will be immediately broadcast to all the clients. Clients can obtain information on the most updated game world and interact in real-time manner.

3.4.2 Dynamic

The system architecture allows the clients join the game dynamically without creating any session or need to wait for other players. The master server provides information for the client to find out the game servers on the Internet and once a game server is available, the client can join the game immediately.

3.4.3 Modularized Architecture

The game engine is design as a module within the system. Also, the game engine itself is composed of several modules that co-operate to provide all the function. This design allows more modules to be added to the system easily.

Besides, the modularized architecture allows the developer to replace any modules as long as the API remains the same.

3.4.4 Scalable

Since the system uses the multi-server architecture, it allows more and more game servers to be dynamically added to the game server group. Therefore, the scale of the game can be extended unlimitedly in theory. If more clients join the game, more servers can be use to handle the clients. On the other hand, when the number of clients in the game decreases, some servers can be terminated and all its clients can be transferred to other servers. With this feature, the game can be much more scalable.

3.4.5 Extensible

Extension can be added to the game platform as required. Since the game uses modularized architecture, many different extensions can be added to the game by inserting new modules to the current game platform. As the game may evolve over time, this can be achieved only when the game platform allow different extension to be added.

3.4.6 Flexible Application Layer Protocol

The application layer protocol is described in the diagram below:

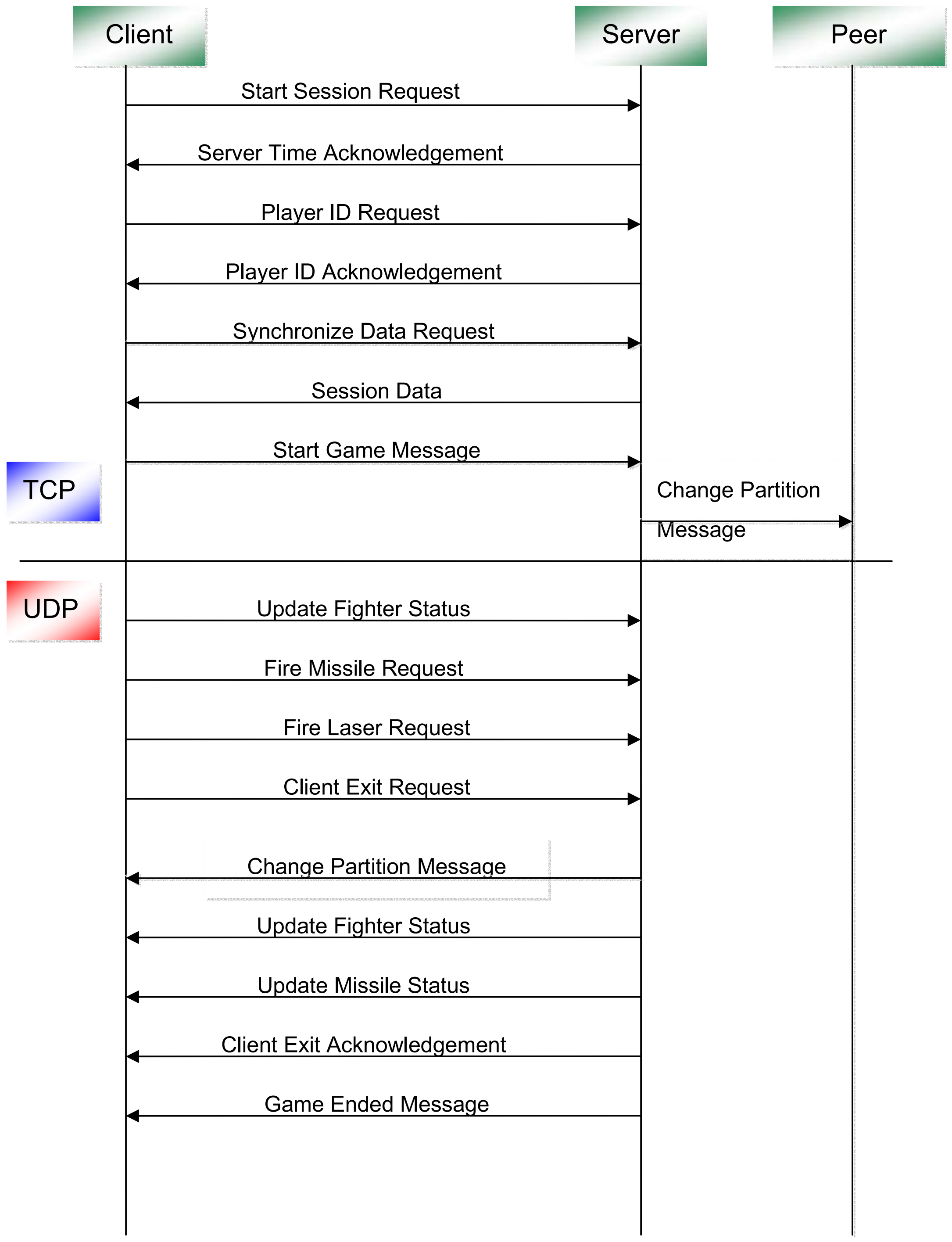


Fig. 3.5 application layer protocol

The application protocol consists of different game message types:

Server Time Request: Client asks the server for the server system time

Server Time Acknowledgement: Server sends back the system time to the client

Player ID Request: Client asks the server for the player ID that uniquely identifies the player in the game world.

Player ID Acknowledgement: Server sends the generated player ID to the client.

Synchronize Data Request: Client asks for the server to send the current game state information.

Synchronize Data Acknowledgement: Server sends the most current game state information to the client. The information includes the fighter and missile information.

Start Game Message: After all the game state information has sent to the client, the server will send the start game message to the client.

Update Fighter Status: The client sends the most updated status of the fighter to the server. Also, the server will send the updated status of other fighters to each fighter.

Update Missile Status: The server sends the most updated position of the missile to the client.

Fire Missile Request: When the client fires a missile, it will send a fire missile request to the server.

Fire Laser Request: When the client fires a laser, it will send a fire laser message to the server.

Client Exit Request: When a client wants to quit the game explicitly, it will send a client exit request to the server.

Client Exit Acknowledgement: After the server receives the client exit request, the server will send back a client exit acknowledgement to the requested client as well as the other clients

Game End Message: When the last game server on the internet going to terminate, it will send a game end message to all the clients so that all clients will terminate.

Change Partition Message: When a client changes from one partition to another, the server of the original partition will send a change partition message to all the clients within the same partition. Also, it will send a change partition message to the server of new partition.

The application layer protocol can be extended by adding different message type and corresponding handler to the main game loop. New message type can be defined as long as it is compatible to the message format that is defined in the Network Engine. Therefore, the application protocol is very flexible and more message types can be added if more features need to be supported.

3.5 Problems and Solutions

3.5.1 Multi-server architecture

In the early design stage, the single server model is the proposed architecture for the game system. However, if only one server on the Internet can support the game, there will be several drawbacks:

Network traffic
If only one server is used to handle all the clients at the same time, the network traffic at the server side will be very heavy. The network traffic of single and multi-server architecture will be presented in the Evaluation section below.

Server Loading
Since the server is responsible for control the whole game world and the server needs to process all the client messages. All the computation tasks except the graphics rending are carried out at the server side. If only one single server is used to handle all the clients, the loading of that server will be very high.

Scalability

With only one server, the number of clients allowed to join the game is limited since the computational power and the network bandwidth is limited. Therefore, with only one single server, the scalability of the game will be restricted.

With the multi-server architecture, the drawbacks described above will not happen. Actually, the multi-server architecture is a typical kind of distributed computing. The distribution of tasks over a group of game servers can:

Distribute the Network Traffic

Since a client will only communicate with one game server at a time, this means the clients are divided into group and each group will be handled by one game server. Then, the network traffic will also be divided among the game server group and will not be as intensive as the single server architecture.

Although the multi-server architecture will introduce the inter-server network traffic, the inter-server traffic is very small as it only consist of the change partition messages and the change partition message will not be as frequent as the update position messages of the fighters and missiles. Therefore, the multi-server architecture can reduce the network traffic demand for the server.

Fewer Requirements on the Servers

Since the computation is distributed over a group of game servers, each server needs to handle the clients belonging to it and the number of messages needed to be handled will be much smaller compared with the single server case. Therefore, the computational power requirement is less for the server.

Distributed Game State

Each server need to keep a fraction of the whole game state. In order to obtain the whole game state, all the game states on the servers need to be combined. This is the essential reason for inter-server messages.

More Scalable

The multi-server architecture allows new game servers to join the server group dynamically without termination of all current servers. That means infinite servers can be added to the server group and the game can scale to support infinite number of clients theoretically. However, in the single server architecture, there is a limit on the number of clients.

Avoid Single Point of Failure

If the server in the single server architecture failed, the whole game system will fail. In contrast, the multi-server system will not fail if only one of the servers is down.

However, one may argue that if the master server on the Internet is down, the whole game will also down. Actually, if the master server on the Internet is down, the game will not terminate. The only consequence is that no new client can join the game. Since the master server only perform the initial negotiation between the server group and the client, once the game is started, the master server will not involve in the communication between the clients and the game servers. Therefore, if the master server is down, the current game can still continue.

Efficient Static Partition

With the multi-server architecture, the static partition can be done efficiently. Each static partition can be handled by one game server and more detail description on the static partition will be given in section 4.5. All the requirements are examined in real tests with PCs in LG103, CYC.

	Single-Server	Multi-Server
Network Traffic	> 5Mbps	> 1.5Mbps
Server Loading	High	Low
Server Requirement	Powerful PC	Ordinary PC
Scalability	No	Support infinite clients
Game State	Centralized	Distributed
Static Partitioning	No Effect	Reduce more than 50% traffic
Inter-server messages	No	Within 1Kbps
Availability	Single point of failure	Better

Fig. 3.6 Comparison between single-server and multi-server architecture

To conclude, since the multi-server can overcome the drawbacks in the single sever architecture, the multi-server architecture is chosen to be the system architecture.

3.5.2 How to allow dynamic joining of the game

When designing the multi-server architecture, one of the problems is how to let the clients join the game dynamically. Since most real-time Internet games require the users to create and join a session before they can play the game.

In order to allow the players to join the game directly without the session, the server need to generate an initial position for the client that is not occupied by the other clients. After the server find out such a position, the client can start the game at that position. Before the player can start playing the game, the client need to synchronize with the server so that the client can get the most current game state and start the game. These message exchanges have already been captured by the application layer protocol design.

3.5.3 How to increase the scalability and extensibility

Scalability and extensibility are very important features of this game. Therefore, some techniques have been employed to increase the scalability and extensibility of the game platform.

The multi-server architecture contributes most to the scalability of the system. This allows the system to be changed to different scale dynamically. For the extensibility, a flexible application layer protocol has been used and also the modularized design of the system contributes to the extensibility of the whole system.

3.5.4 Application Layer Protocol Design

When designing the application layer protocol, one major goal to achieve is flexibility. The application layer protocol is used to define the message exchange between the client and servers. To increase the flexibility of the application protocol, the protocol should be allowed to change easily. More message types can be added and the sequence of the message can change easily. In order to support this, the underlining network layer needs to have the message encoding and decoding. Since different message type will contain different information, the encoder and decoder need to handle all the message type and transmit the message over the internet.

Also, the handler for the new message type should be added easily to the game engine to handle the new message type. Then the application protocol can be extended without any difficulties.

3.6 Current Implementation

Currently, almost all the features talked about in this section are fully built into the game, including the multi-server and multi-client support, the application protocol, dynamical joining of the game, and the modules used in the client and server.

There are client-server communication and inter-server communication. The clients can be handled by different servers and will be transferred to other servers depending on their real time status, and the server will negotiate the migration of clients to each other, which is transparent to the client and won't affect the effect at clients.

Because of the limited time, the only unfinished part of the system is the master server. Instead, hard coding is used temporarily. However, full flexibility is left for a master server and adding it is not difficult.

3.7 Evaluation

3.7.1 Objective

This simulation is to test the possible work load for single server and multi servers.

3.7.2 Assumptions

1. The clients generated are not sending the firing messages. This is because the count of firing messages is much smaller than the traffic flow of position updating messages.

To cope with the real time requirement, the position of each client is updated 25 times per second. No matter how fast a person being is, it is very unlikely that he/she will send the firing messages continuously and with the same frequency of the position update message. Thus the real traffic flow is at most twice of the testing flow got in the experiment.

2. The removal of 3D graphics at the load generator will not decrease the network flow. In fact, this is only possible to increase network flow since the burden of the client is reduced. As the simulation here is mainly carried about the server ability and the network flow, it is all right to remove 3D graphics part at the client side.

3. With multiple clients simulated on one single PC, the clients are able to perform at the normal speed. In another word, the clients simulated will generate the same load for the servers.

This is justifiable. The load generator is loaded as 10 network thread and 10 client fighters per generator (more loading per generator is possible, but may lower down the speed of the client, causing inaccurate result). With the same network flow as 10 client fighters, the server will carry out the same amount of work as there are 10 clients.

3.7.3 Environment

All the server and client machines are of the following hardware configuration.

Basic Environment - Hardware		
Item	**Name**	**Additional Info**
CPU	Intel P4 1.4G	256K L2 Cache
Mainboard	DMI	400MHz
Memory	256MB DDR	
Network	3COM Ethernet 100M	
Graphics Card	GeForce 2 GTS	32MB VRAM
Basic Environment - Software		
Item	**Name**	**Additional Info**
Operating System	Windows XP Pro	Build 2600
IDE	Visual C++ .NET	V 7.0
Network Monitor	LinkFerret Network Monitor	V 3.07

Programs (client and server) are compiled with all optimization and the performance is maximized. All threads are running at the normal frequency and all tasks at the server side are performed.

Each client is running a load generator. The load generator will generate 10 clients connecting to the server. Each server is running an instance of the server program.

Two experiments are carried out. Firstly we use 40 clients connecting to a single server, and then 40 clients connecting to 4 servers with inter-server interaction. In the 2 experiments, the data monitored about the CPU and network usage are listed in the following table.

Client Generator and Real Server Testing			
Client & Server Count			
Server Number			1
Client Number			40
Clients per Server			40
Performance Measurement			
	Client		Server
CPU	100%		100%
Network Usage	51.23Kbps		2590Kbps
Client-Server	51.23Kbps		
Inter-Server			0

Client Generator and Real Server Testing				
Client & Server Count				
Server Number				4
Client Number				40
Clients per Server				10
Performance Measurement				
		Client		Server
CPU		100%		100%
Network Usage		54.028Kbps		777.5Kbps
	Client-Server	54.028Kbps		
	Inter-Server			0.0237Kbps

3.7.4 Result Data Analysis

According to the data collected above, the multi-server architecture can reduce the network requirement and computation amount at each server, which is a basic advantage of distributed computing. The network usage at the client side basically remains as constant, while the inter-server message is so small and can be neglected. Thus using the multi-server architecture is beneficial.

3.8 Future Planning

3.8.1 Cluster Architecture

The multi-server architecture can be extended to the cluster architecture. One logical server can be run on the cluster and actually each machine within the cluster is a game server. Therefore, the cluster is actually a group of game servers. The advantage of the cluster architecture is that the inter-communication delay between the game servers can be minimized. Also, with the cluster architecture, the game state can be centralized to one physical space and it is much easy to maintain the game state and make sure the game state is in consistence state.

3.8.2 Flexible and extensible platform

The flexibility and extensibility of the game platform can be further enhanced so that the game can evolve over time.

3.9 Conclusion

To conclude, the game engine cooperates with other modules in the system to achieve the overall functionalities. Also, the game engine incorporates the flexibility and extensibility so that it can be an excellent platform for the game to develop continuously.

Chapter 4 Network Engine

4.1 Overview

Network engine's main function is to handle all the underlying networking issues.

4.2 Architecture

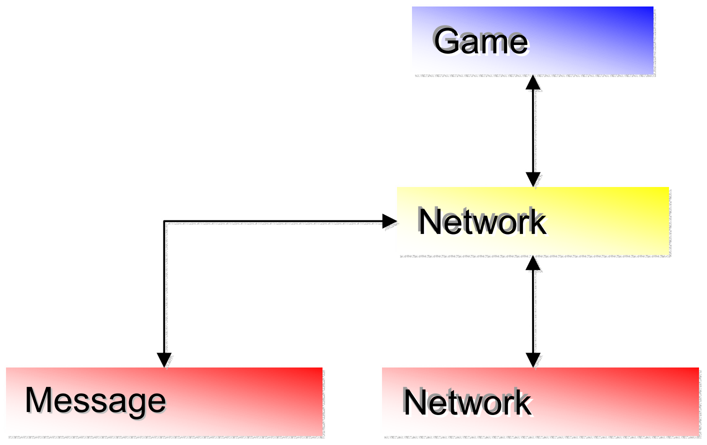


Fig. 4.1 interactions between network engine and other components

As shown above, the architecture of the network engine is quite simple and clear. It is defined in such a way that both client and server can use this architecture for sending and receiving data.

Network engine functions by delegating request from the Game Engine to the main class NetworkEngine and this class utilizes the NetworkResources and MessageSerializer classes.

NetworkEngine provides an interface for Game Engine to call for services. Its functions include:
Sending and receiving of data to and from the internet.
Setting up the connections between clients and server and the connections between other peer servers.

NetworkResources class acts as a data store and store the data received from the network and the connection specific objects.
MessageSerializer class is responsible for encoding or decoding the message received from the internet to the format that can be understood by the network engine.

4.3 Features

4.3.1 TCP Communication

Two types of messages are sent and received using TCP. They are game initialization messages and inter-server messages.

During game initialization, all settings of the game must be sent to all clients and these settings should be synchronized among all clients and servers. Thus TCP is a natural choice for sending such type of messages as no message loss can be tolerated in this stage.

Now let us consider the communication between servers. The only communication between the servers is the client information when a client changes from one partition to another. Obviously, this type of message must be received correctly without loss; otherwise a client may suddenly disappear during the change of partition. Therefore, again, TCP is chosen to be the communication protocol for this type of message.

4.3.2 UDP Communication

During the game, there are many update messages being sent and received between clients and servers. These update messages, however, are tolerable against loss. For example, an update message carrying the position update of a client can be lost because a later message will arrive within several milliseconds. Therefore, the change in position is not too sensitive and hence we use UDP for this type of communication.

4.3.3 Redundant Message avoid Data Loss

UDP is an unreliable protocol, which causes message lost, so redundancy is used for reliability. The idea is simple, for every message we sent out, we embed 2 previous commands into the current message. In such a way, even when some messages are lost during transmission, the receiver can receive the next message and see if the previous commands embedded in the message have been executed. If the commands were executed, they can be simply discarded, otherwise, the commands are executed.

4.3.4 Multi-thread support

There are multiple threads implemented to aid the work of the network engine.

For the server, one thread is used for continuously accept clients to join the game. Another thread is used to periodically receive messages from accepted clients.

To implement the multi-server approach, two more threads are used to maintain the connection with other peer servers and receiving messages from them respectively.

Finally, one more thread is implemented to receive the UDP messages from clients. This thread is event-based.

4.3.5 Fixed-Format Message for Internet Transmission

We have defined a message scheme such that for each message type, the content of the messages are well-defined. For example, for the update messages, the fields 7 – 15 in the message are reserved for holding the eye-point, head vector and up vector of the client. In such a way, both client and server can have a fixed way for communication. (For details of the universal message scheme, see Appendix A4)

4.3.6 Message Encoding and Decoding

The messages that we send to and receive from the network are actually strings. To ensure that both client and server understand the meaning of the string, we have implemented a simple encoding and decoding scheme.

Whenever the game engine invokes the network engine to send a message, the messages, which are initially in our self-defined format (a range of fields defining the receiver of the message, message type, message content), are encoded into a string. After that the strings are sent to the network by the network engine.

Whenever the game engine needs to receive a message, the messages, which are initially strings, are decoded to our self-defined format. After that the message can be processed by the game engine.

4.4 Advanced Topic: Synchronization

4.4.1 Overview

The importance of synchronization

In the real-time internet game, synchronization is one of the important aspects. Since the interaction between the players is in a real-time manner, if all the players are not in sync, some strange behavior may arise. For example, if a client 1 fire a laser and the laser must shoot the target in his view, then the client 2 notices this and issues several down commands to escape from the laser. However, due to the network delay, the message form the client 1 arrives to the server late and the client 2's messages arrive first. Then the client 1 cannot destroy client 2. This situation is illustrated in the following figure.

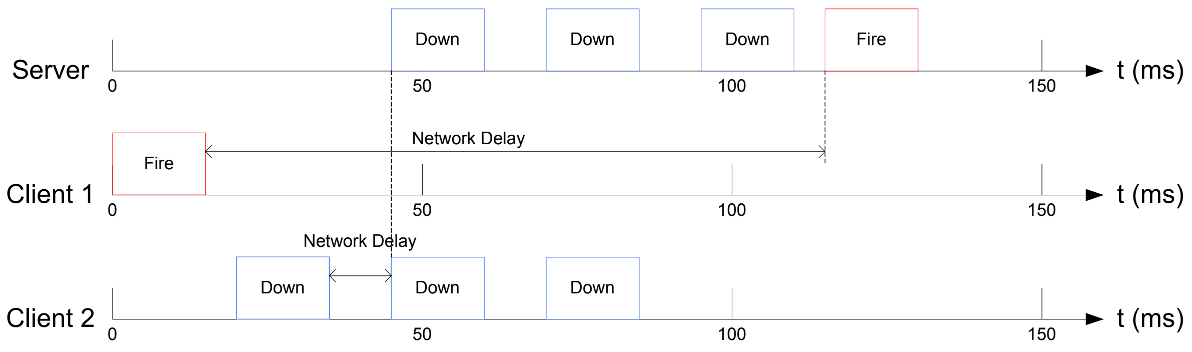


Fig. 4.2 the scenario showing the importance of synchronization

In order to avoid the situation above, synchronization is needed to make sure the game messages are processed in order.

Synchronization mechanism

In general, synchronization will be mainly performed at the server side. The synchronization mechanism used in the online real-time game will maintain a database which keeps several snap shots of the game state at different time. Also, it will keep a list of messages received from the client so that all the messages can be processed again after the roll-back of the game state.

The synchronization mechanism used in the game use the time-stamping, time-framing and game state roll-back techniques to achieve the synchronization. The basic synchronization mechanism is shown in the figure below:

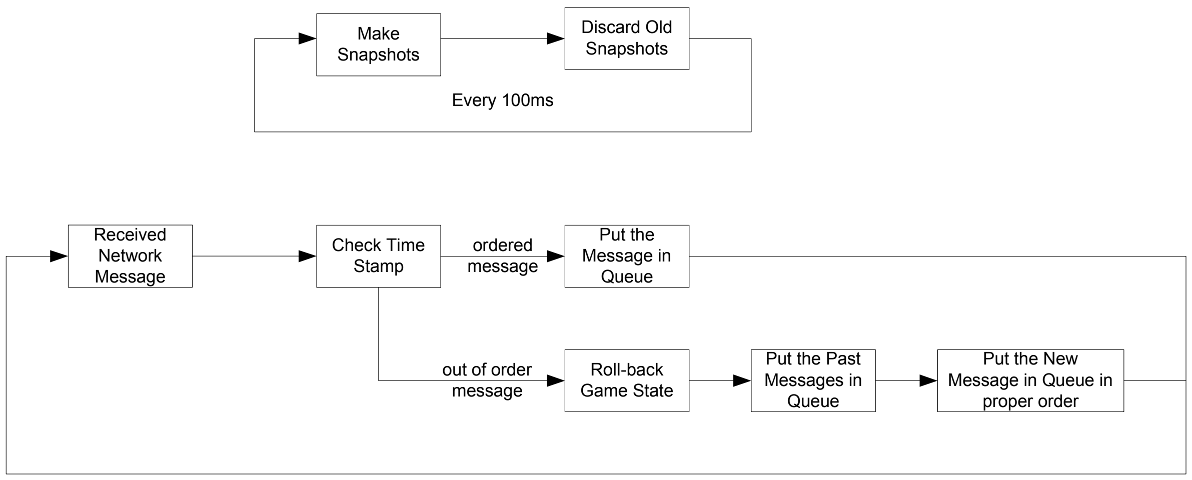


Fig. 4.3 The basic synchronization mechanism

4.4.2 Features

4.4.2.1 Message Classification

All the game messages are classified into two main categories: Critical and Non-critical message. The non-critical message will not be handled by the synchronizer since they will not affect the consistence of the game state. If those messages are late or missed, they will be ignored and will not be processed. The example of the non-critical message is the update fighter position message.

For the critical messages, if their order is reversed or the messages are missed, it will affect the consistence of the game state and cause strange behavior. Therefore, the synchronizer will re-arrange the critical game messages in their sending order and roll-back the game state. After the game state has been rolled back, all the game messages will be processed again.

4.4.2.2 Time Stamping

At the client side, before any game message is send out, a time stamp will be attached with the game message. The time stamp will be used by the server to determine the order of the game messages.

4.4.2.3 Time Framing

The time at the server side will be divided into frames. Each frame will have a length of 100ms. At the end of each frame, a snap shot on the Data Store will be captured. Those snap shots will have a life time of 200ms and they will be used when there are out-of-order messages and the game state needs to be rolled back.

Also, when a game message is received from the client, the game message will be stored and a list of in-order game messages is maintained in the synchronizer. It will be used after the roll back of the game state. All the game messages will be re-executed and bring the game state to consistence state.

4.4.2.4 Game State Roll-back

If there is an out-of-order critical message, the synchronizer will roll back the game state by using the snap shot stored every 100ms. Depend on the delay of the message, different snap shot of the Data Store will be used to replace the current inconsistent Data Store.

If the delay is within 100ms, the Data Store100ms before will be used to cover the current Data Store. If the delay of the message is within 100~200ms, the Data Store 200ms before will be used. Finally, if the delay is more than 200ms, the message will still be handled by the server. However, the behavior or effect of those messages will not be guaranteed to be correct. Therefore, the slow clients may experience delay in the effect of their actions and some strange behavior may happen.

4.4.2.5 Maximum Delay Allowed

As mentioned above, the message with delay more than 200ms second will not be guaranteed to be correct. That means the maximum allowable delay of the game message is 200ms. If the delay is greater than 200ms, the order of the messages may not be correct.

Actually this maximum value can be increased by increasing the number of data store copies stored in the synchronizer. However, this is limited by the computational power of the CPU and the actual delay of the network.

4.4.3 Problems and Solutions

4.4.3.1 Problem about the time stamping

Since the time at the servers and the clients is different, some mechanism is needed to synchronize the server time with the client time before the game start.

The following steps will be carried out to synchronize the server time and client time.

1. Client record the current time t_i.
2. Client send a time request message to server
3. Server send back the current time of the server t_s to the client
4. Client receive the server time and record the current time t_j
5. Assume the processing time is negligible compare with the network delay, client calculate the time difference due to network delay $d = (t_i – t_j)/2$
6. Client calculate the approximate time difference between the client and server $t_d = t_j – (t_s + d)$

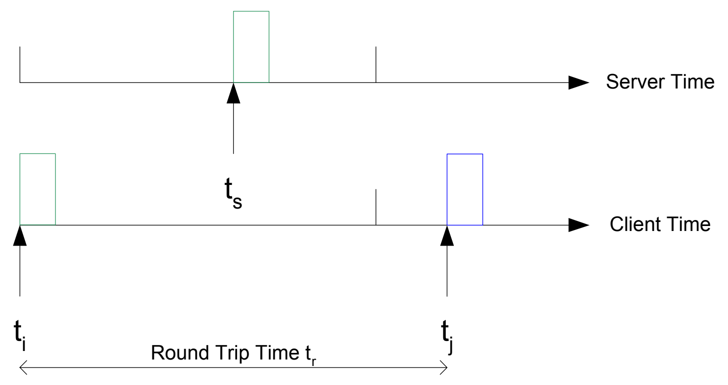


Fig. 4.4 synchronize the server time and client time

With the time different between the client and the server, the client will add this difference to the time stamp so that the client time and server time can be in sync.

4.4.3.2 How to avoid un-necessary synchronization

Since the synchronization requires the roll back of the game state and re-executes all the game messages, if too frequent synchronization is performed, it will waste a lot of processing power. Besides, there are some game message types that the message can be missed without causing inconsistence in the game state.

Therefore, the game message will be classified into critical or non-critical message. Only the out of order or missed critical message will trigger the synchronization. The non-critical message will not. Then the un-necessary synchronization can be avoided.

4.4.3.3 How to minimize the amount of data backup

In the synchronization, it will keep backups of the Data Store for the roll back operation. However, how many backups are enough for this purpose??

Actually, a trade off between the memory space and the precision of the synchronization needs to be considered. If more backup is kept, the synchronization can be done using a smaller time frame. Also, with a smaller time frame, the number of game message needed to be re-executed will be also smaller. However, this will increase the demand for the memory space.

Since the size of the Data Store will become bigger and bigger when more clients join the game, it is not possible to keep many such a huge Data Store in the memory. As a result, only two Data Store backups will be kept in the memory at any instant. This can save the memory space and perform the synchronization with a reasonable time frame of 100ms.

4.4.3.4 How to classify different message type

In classifying the message into critical and non-critical category, we need to consider how the out of order of missed messages will affect the games state. The following table summarizes the reason to classify the message type into each category:

Critical Message	Reason
Missile Fire	The out of order missiles may cause a client to be destroyed wrongly.
Laser Fire	The out of order missiles may cause a client to be destroyed wrongly.
Client Exit / Destroy	If the server miss this message, the exited client may still remain in the 3D game environment
Change Partition	If the server miss this message, the client may loss the contact with the servers since the new server does not know a client has join its partition
Add Client	If the server or client miss this message, the new client may not join the game correctly
Non-Critical Message	
Update Fighter Position	If the server misses one or two update fighter position, it will not cause serious inconsistence since the next update fighter position will review the current position of the fighter.
Update Missile Position	If the server misses one or two update missile position, it will not cause serious inconsistence since the next update missile position will review the current position of the missile

4.4.4 Evaluation

4.4.4.1 Current Implementation

The current implementation fully deployed all the features talked in the above section, with flexibility in several parameters like the number of data copies maintained, the synchronization interval, and so on. Two threads are implemented, with one in charge of taking snap shots and the other in charge of actually sorting the messages and process them. The synchronization threads are of high frequency to ensure the real-time effect is achieved.

4.4.4.2 Objective

This simulation is to find the number of delayed packets transmitted over the network.

4.4.4.3 Assumptions

1. Only the critical messages will be monitored by the synchronizer and the order of those messages will be guaranteed. On the other hand, the non-critical messages will not be monitored by the synchronizer since they will not affect the consistency of the game state.

The critical messages include the firing messages change partition messages and client exit messages.

2. Assume the underlining system will utilize approximate 600ms to process and send out the message. Therefore, if a message's time stamp has the different more than 600ms, then it will be regarded as delayed messages.

4.4.4.4 Environment

There are 1 client and 1 server. All the clients and servers are of the following hardware equipment.

CPU: P4 1.4G
RAM: 256MB DDR RAM
Network Connection: 100M Ethernet
Network Condition: With large amount of network traffic
Programs (client and server) are compiled with all optimization and the performance is maximized. All threads are running at the normal frequency and all tasks at the server side are performed.

4.4.4.5 Result Data and Analysis

Following is the table showing number of messages delayed due to the network congestion.

Test	1	2	3	4
No. of Fire Messages	20	20	40	40
No. of Delayed Messages	13	12	25	22
Percentage loss	65%	63.25%	62.5%	55%

Form the result above, the average percentage of delayed messages is 61.4375%. The percentage of delayed message is over 50% when there is large amount of traffic. Therefore, if no synchronization is used to guarantee the order of the fire message, this may put the game into inconsistence state.

Given this large proportion of delayed messages, the synchronization requires a lot of computational power. Currently each invoking of the synchronization threads (including the back up thread and the synchronization thread) will cost as much computational power as the collision detection module. However, in the tests carried out, the required computational power can be provided by a normal P4 1.xG CPU. Thus the requirement is still reasonable.

4.4.5 Future Planning

4.4.5.1 Advance Game State Backup

Since the backups of the game state will consume a lot of memory space, therefore, some techniques can be employed to compress the backups to save memory. The data structure used to store the dynamic entities of the game can be redesigned.

If the backups can consume less memory, more backups can be stored in the memory and we can have a smaller time frame for the synchronization. Since the size of the time frame can be decreased, the synchronization can be more accurate and the number of game message needed to be re-executed will be smaller.

4.5 Advanced Topic: Partition Techniques

4.5.1 Overview

During the game, update messages are sent from clients to the server quite frequently. For each update message, the server has to broadcast the update to all clients within the same server. As the number of players hosted by a particular server increases, the network traffic between clients and that server also increases and significant delay will be experienced by clients playing the game. Are there any methods to reduce such kind of network traffic and improve the performance?

One of the methods is partitioning. The idea behind is to divide the map into smaller segments called partitions and each client only know the existence of other clients, who are within the same partition. In other words, update messages about a client in a particular partition will only be sent to other clients within the same partition, but not other partitions. In such a way, the network traffic between clients and server is reduced, since for each client update, the number of update messages that need to be sent is reduced.

In the following paragraphs, the features of partitioning are illustrated. The problems encountered when implementing partitioning is discussed. The performance of the partitioning techniques is evaluated, in particular, how can network traffic be reduced and by what amount. Finally, some possible future extensions to the current approach are discussed.

4.5.2 Features

4.5.2.1 Characteristics of spatial partitioning approaches

Criteria	Static Partitioning	Dynamic Partitioning
Complexity	Easy	Complex
Flexibility	Inflexible	Flexible
Number of partitions	Fixed	Varied
Existence of partitions	Permanent	Temporary
Scalability	Low	High

Based on the above comparisons, we conclude that both partitioning schemes could be applied, but at different stages of the game. Before the start of the game, we know the number of clients exactly and hence we can easily divide the virtual world into static partitions in such a way that the workload is balanced among the servers. However, during the game, when there are more and more clients hosted by one server, we have to place them in different partitions to reduce network traffic. However, static partitioning won’t work since the newly created partitions should be deleted later when the number of clients in that partition reduced to normal. Therefore, we should apply dynamic partitioning here.

4.5.2.2 Static partitioning

Design

In static partitioning scheme, the virtual world is divided into several smaller partitions. The word “static” here means that these partitions will not be modified or destroyed during the game. The number of static partitions depends on the number of clients playing the game and also the number of clients that can simultaneously stay in one partition without experiencing serious network delay. Our goal is to allow the clients to be evenly distributed among all partitions initially so that the workload can be balanced among the servers.

Implementation

In our implementation, each server will hold one static partition and there will be four static partitions in total. However, this configuration is for demonstration purpose only, each server can actually hold more than one partition. The servers can communicate with each other at a high speed because they are placed in a Local Area Network (LAN) with a bandwidth of at least 10Mbps with the Ethernet.

After a client connected to one of the servers, the server will send all the information of clients within the partition to the connected clients before starting the game so that they can know about the information of other clients within the same partition.

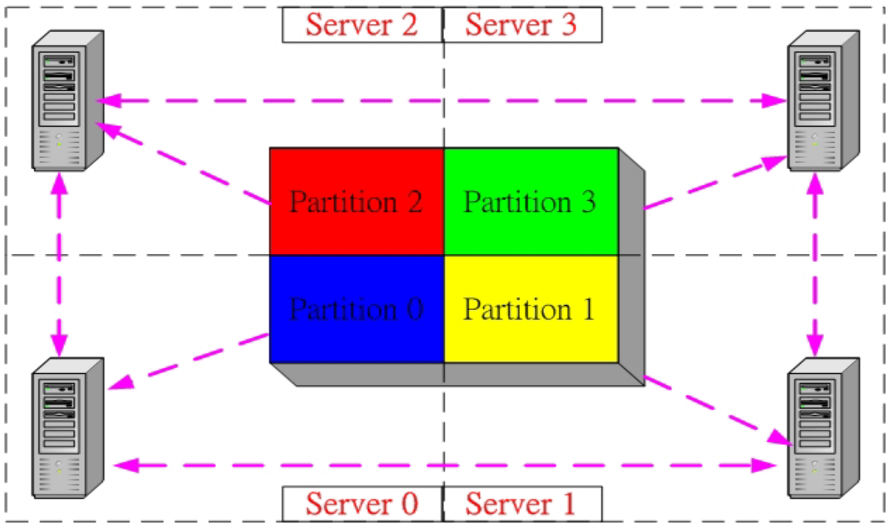


Figure 4.5 Initial configurations of the servers and partitions

4.5.2.3 Interest filtering

One of the features of the partitioning schemes is the introduction of interest filtering. Within a particular partition, clients are interested to information about other clients within the same partition as itself. For example, they need to know the positions of other clients within the same partition, whether another client is dead or alive. Anything other than these may not be of any interest to the client and by no means need to be stored in clients.

Partitioning helps interest filtering by grouping clients into clusters such that each cluster is represented as a partition and hosted by a server. In such a way, whenever the server has to send information to a client, only information within the same partition as that client needed to be sent. As a result, interest filtering is achieved and clients do not receive any information about clients from rest of the virtual world.

4.5.2.4 Overlapping region of interest

When clients are close to the boundary of its current partition, they should be able to see other clients that are close to the boundary, but within other partitions. To handle this, we defined a region near the boundaries of partitions such that information about clients within this region is known to servers that are hosting the partitions with the boundary. The length of the region is equal to the maximum visible range of the fighters so as to ensure that clients within different nearby partitions can see each other.

When a client enters this region, the server hosting this client will send the information of this client to another server that is hosting the partition on the other side of the boundary. Both servers keep the information of this client until the client leave the region. At that time only one server hold the information of this client.

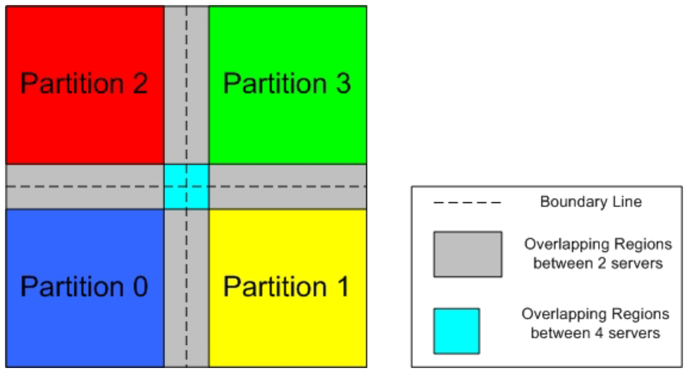

Partition 2
Partition 3
Partition 0
Partition 1
Boundary Line
Overlapping Regions between 2 servers
Overlapping Regions between 4 servers

Fig. 4.6 Overlapping regions of interest

As shown above, the region can exist in two forms:

(1) shared between two servers

This is the case to handle a client changes partition along horizontal or vertical direction, for example, from Partition 2 to 3. In this case, both servers on either side of the boundary should have the information of the clients within the region.

(2) shared between four servers
This is the case to handle those client changes partition along the horizontal, vertical as well as diagonal direction, for example, from Partition 2 to 1. Since the small center region is the common region owned by the four servers, all four servers should have the information of the clients within this region.

4.5.3 Problems and Solutions

4.5.3.1 Exchange of client information among servers

With partitioning, when one fighter tries to move from one partition to another partition, this change of partition must be known by all clients within the two partitions and hence we should broadcast the change partition message to all related clients. However, if we broadcast the change partition message at the time the fighter crosses the partitions, it will be too late since the processing of this change of partition takes time. Clients in the new partition may not be able to know the position of the new client and collision detection cannot be done correctly.

The solution to this problem is to use the overlapping region of interest approach mentioned above.

4.5.4 Evaluation

4.5.4.1 Amount of network traffic

The following is the evaluation of the importance of partitioning technique to amount of network traffic. The evaluation is done by comparing the network traffic of server when using partitioning techniques against that without using partitioning techniques:

Assumptions

1. All clients and servers have same background traffic.

Environment

All the clients and servers are of the following hardware equipment.

CPU: P4 1.4G
RAM: 256MB DDR RAM
Network Connection: 100M Ethernet
Network Monitor: Link Ferret Monitor v5.2

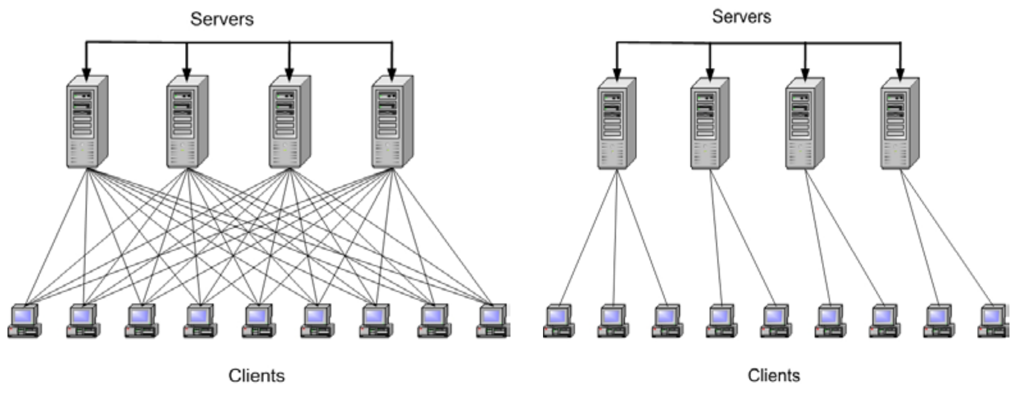
The clients are generated by four load generators and each of which generate 10 clients. The load generator will generate certain number of clients connecting to the server. Each server is running an instance of the server program, and the servers maintain interaction about the partition changing of the clients. The position of each client is updated 25 times per second, and the servers collect all the messages and distribute the updated client information to the other clients.

Apart from the client generator and server, each PC is running the Link Ferret network monitor to monitor the network flow.
There are 40 clients and 1 server for testing network traffic of the server without partitioning. There are 40 clients and 4 servers for testing network traffic of the server with partitioning.

Result Data and Analysis
Background traffic = 349838 bytes per 300 seconds

	Number of clients	Number of server	Duration (second)	Inter-server Traffic (Kbps)	Client-Server Traffic (Kbps)	Traffic per server (Kbps)
Without partitioning	40	1	300	0	2590	2590
With partitioning	40	4	300	0.0237	777.5	777.5237

From the above result, we can conclude that partitioning help reduce the network traffic flowing through the servers with the network traffic reduced by more than half of that without partitioning.



(a) (b)

Fig. 4.8 (a) Network traffic without using partitioning
(b) Network traffic using partitioning

4.5.4.2 Maximum number of clients supported in each server

Since the bandwidth of the server is limited, we have to calculate the maximum number of clients that can be in a particular server for partitioning to work. We found that the maximum number of clients that can be supported by a server is 40. This is obtained by using more and more clients to connect to a single server until the server has no response.

4.5.4.3 Number of servers required

Based on the above calculations, we can calculate the number of servers required for hosting the whole game given the total number of clients. It turns out that the number of server required can be found by:
N >= P / S where
N = number of servers
P = total number of clients
S = maximum number of clients supported by each server

4.5.5 Future Planning

4.5.5.1 Dynamic partitioning approaches

Tree-based dynamic partitioning

When the number of clients in one partition increases to a level such that the bandwidth requirement of the server exceeds its given bandwidth, which means the messages from clients are more than that can be handled by the server; something has to be done to reduce the network traffic back to an acceptable level. Here comes the dynamic partitioning scheme.

The idea behind dynamic partitioning scheme is to further divide the existing static partitions to even smaller partitions so that all clients only receive messages related to other clients within the same partition and this helps to bring the network traffic to an acceptable level. When the number of clients decreases such that the network traffic becomes normal again, these dynamic partitions are removed.

To implement the dynamic spatial partitioning scheme, we have two major choices (i.e. Octree and Binary Spatial Partitioning (BSP) tree) that are commonly used in the gaming industry, let's have a comparison about them as follows:

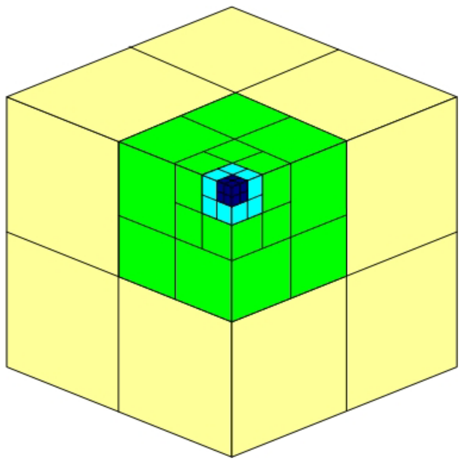
Criteria	Octree	BSP tree
Number of children of each non-leaf node	8	2
Nature of virtual world	Outdoor	Indoor
Searching Efficiency	$O(\log_8 N)$	$O(\log_2 N)$
Implementation complexity	Simple	Complex
Tree Construction Efficiency	Faster	Slower
Shape of partitions	Rectangular	Variable

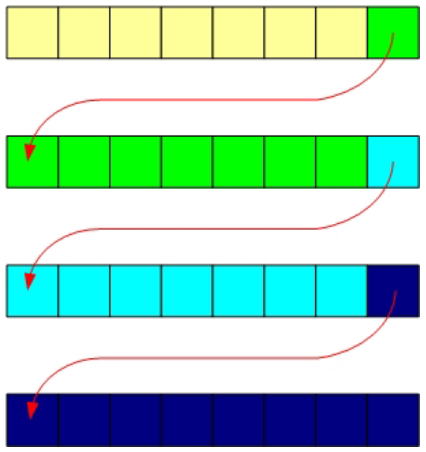
Based on the above comparisons and our game nature, Octree is chosen to implement the dynamic partitioning scheme. The most distinguishing reason is that BSP tree depends on splitting planes to divide the virtual world, which are usually walls or something that separate the virtual world naturally in an indoor environment. However, in our game, there is no wall. Our virtual world is an outdoor environment! Another important reason is that the partitions resulted by using BSP tree are of irregular shape, which makes computation difficult. Last but not least, the implementation for BSP tree is too complicated that the gaming industry spend years of effort to optimize it, and it is thus infeasible for us to implement it in the final year project given the timing constraint.

Let's see how this works by first examining the concept of octree and see how it is used for partitioning.

Octree is a tree data structure with a maximum of eight children for each non-leaf node. Octree is commonly implemented as an array of partitions with a pointer extending from each non-leaf node to its children. Octree is commonly used in computer graphics to do spatial partitioning for collision detection and frustrum culling. Here, however, besides collision detection, we use it for reducing the network traffic between clients and server.

It is used to divide a static partition into eight smaller rectangular shaped partitions. The original static partition is considered to be the root of the octree and the eight newly created partitions are the children of the root node. All clients within the original static partition are allocated to each of these newly created partitions according to their positions. Each node in the octree will hold all information of a partition. When many clients exist in the same partition, that partition will be recursively divided into smaller and smaller partitions until the network traffic can be handled by the original server.





(a) (b)

Fig. 4.6 (a) Idea behind dynamic partitioning using octree
(b) Array-and-pointer representation for octree, each parent links to (at most) eight children

Area-of-interest (AOI) based approach

Area-of-interest is defined as the visible area for the clients along a plane. In other words, it's the area, which covers all objects in the virtual world, that can be viewed by clients and hence client is interested in it. This area can be defined to be of any shapes, for example, circular, rectangular.

Having defined AOI, we can go further to explain how this can be used to implement a dynamic partitioning scheme. The idea is simple. For all clients within the virtual world, those clients with their AOI overlapped are included in the same partition. This is better in the sense that clients within the same partition are tightly related to each other and are more likely to reduce the network traffic involved.

Distance-based approach

In this approach, the distances between any two clients are computed and clients are grouped together in a partition with other clients that are close to them. There are many distance metrics that can be used in this approach, for example, Manhattan distance, Euclidean distance. There are some famous partitioning methods that are originally used in clustering data in databases, for example, k-means. However, details of this algorithm are not discussed here.

4.5.5.2 Dynamic load balancing among servers

Currently, the partitions created by the dynamic partitioning schemes are hosted by the same server as its original partition. This is not good enough and can be improved. The reason behind is that when more and more clients enter a particular partition, the workload is resided mainly in one server and this causes unbalanced load between servers such that the network bandwidth of some servers are overloaded, while other servers' bandwidth is wasted.

Therefore, we propose that there should be dynamic load balancing among servers in the future. This can be done by allocating the created dynamic partitions evenly to all servers such that the workload can be distributed among all servers and produce a better utilization of the network bandwidth of all servers in our multi-server approach.

However, there are still some difficulties to be explored and investigated. First of all, there must be a way to pass these partitions information between servers and the communication between servers must be defined carefully. Secondly, different servers may have different network bandwidth and hence the partition scheme should be finely adjusted to handle this and allocate appropriate number of clients to each server according to their network bandwidth.

4.6 Conclusion

In conclusion, the network engine is able to provide the sending and receiving of messages from the internet and convert them to understandable format for the game engine. Some advanced topics that can improve the performance of the network engine were discussed.

Chapter 5 Graphics Engine

5.1 Overview

The Graphics Engine in the game is in charge of producing the virtual scene for the players. It takes the updated models and other contents from the data store, and utilizes OpenGL to render them to the screen. Due to the real-time requirement and the graphic quality requirement, the models should be flexible and the rendering should rapid. The Graphic Engine realizes this.

5.2 Architecture

The following graph depicts the overall architecture of the Graphics Engine.

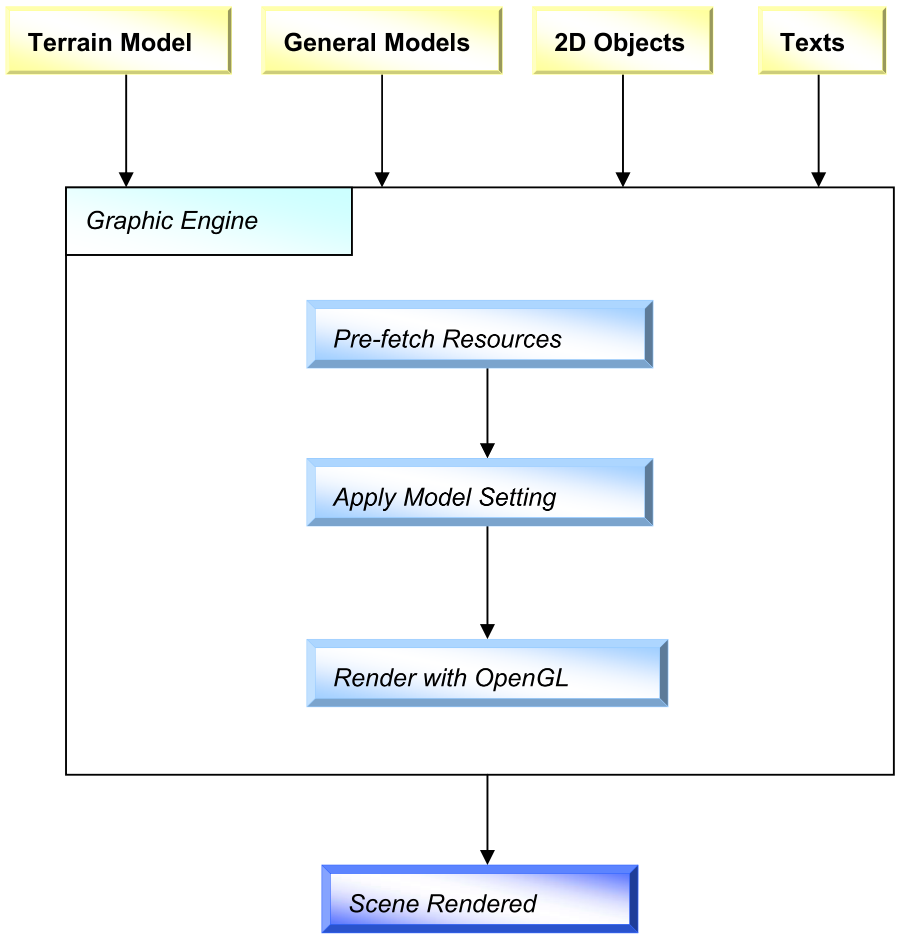


Fig. 5.1 the overall architecture of the Graphics Engine

5.3 Features

- The graphic engine can process 3D models and model groups, 2D bitmap objects as well as texts.
- The generic definition of the 3D model supports most of the rendering options including custom transformation, lighting, coloring, texturing and bounding. Each of the options has several sub-options and can be applied to each 3D model individually. Rendering process is based only on the model, without any hard coding.
- The terrain generation tool is available, which enabling customized fractal terrain generation. Textures will also be generated for the terrains.
- Collision detection, which is also a graphic part, is implemented and can detect collisions among objects in the world. However, this part composes a module in the game logic rather than in the graphic engine.

5.4 Problems and Solutions

5.4.1 Performance

Problem: High performance is critical for the Graphics Engine. Thus it is a great challenge to achieve real-time rendering for client PCs with ordinary graphic cards. In several papers people point out that a refresh rate less than 25 fps will cause perceived delay in updating. Low refresh rate is considered intolerable for players.

Solution: To cope with the real-time requirement, we come up with a series of design and implementation decisions.

- The engine adapts to procedural design rather than following the object-oriented design of remaining parts of the system. The purpose is to follow the pipe-and-filter architecture of OpenGL so that the engine could benefit from the performance optimization provided by OpenGL. Also procedural programming is generally faster than object oriented programming.

- Object culling is applied for complex models. For the complex models such as terrains, culling is applied before actual rendering to reduce the polygons to be rendered as much as possible.

- OpenGL acceleration techniques are widely applied. Recommended implementations such as display list and client state arrays are used, and the calculations are delegated to OpenGL system as much as possible to benefit from the hardware accelerating calculation.

- All matrix calculations are optimized by scientific computing techniques. Doing this further push up the speed as well as accuracy of calculation.

Achievement: The current Graphics Engine demonstrates very high performance. Tests have been organized with the following configurations

Hardware environment (typical configuration for home PCs):
Pentium IV 1.4G, GeForce2 GTS (32MB), 256MB RAM

Software Environment and Parameters:
Windows XP Pro, 200,000 polygons, 512 by 512 BMP texture, game window size 640 * 480, all effects open (including lighting, coloring, material setting, normal vectors, customized transformation, fog, mipmap).

In these tests the Graphics Engine achieved 25~30 fps without any difficulty, which is assumed to be a satisfactory performance.

5.4.2 Flexibility

Problem: Great flexibility must be present for the game, since one of the requirements of the system is the ability to take customized models as the system models.

Solution: Graphics Engine must provide a wide range of modeling options and have a standard, simple and common representation for all models.

- The Graphics Engine data representation is greatly simplified. Only

3 types of objects are required for the Graphics Engine: Model/ModelGroup, BitmapObject, and normal text string (the terrain models are subclass of models). With no game logic specific objects, the Graphics Engine maximizes the extensibility and portability, while minimizes the dependency of render engine on other parts of the system. Any Models and ModelGroups parsed from standard VRML files are guaranteed to be rendered by the engine.

- All possibly desired options are fully provided. Texture, colour, lighting, transformation modes can be applied to each modelgroup and model individually. Especially, the following advanced options are available for each model:
  - Multiple color mode
  - Double texture mode
  - 3 types of transformation mode
  - Bounding box/sphere
  - Height map and culling (for terrain models)

- The terrain models can be generated by the user using the tool provided. As the only restricted object, the terrain needs special information to perform partition and culling. A special tool is provided for generating it to minimize the restriction. For details please refer to the advanced part.

Finally, by combining Models into ModelGroups to form basic unit of 3D rendering, the flexibility is further improved.

Achievement: With the standard VRML and XML files used, great simplicity and flexibility are provided to users who want to construct their own models. Terrains are written as binary model files from the tools. All possible objects can be constructed simply by retrieving the information in VRML from 3DS MAX, XML from XML Spy, and the binary model files output by the terrain generation tool. It is even possible to provide dynamic shaped object with different look for each part (such as planes that can change into robots).

5.4.3 Resource saving

Problem: The hardware requirement will greatly affect the popularity of the game. The assumption that clients are ordinary PCs requires the whole system taking small amount of RAM and CPU cycles. The demanding task of handling multiple sessions containing multiple players also requires we save resources at the server side.

Solution: Resource saving is mainly achieved by reuse and caching. Models and textures are loaded only once and then cached. Reuse of these objects could then be achieved. By separating game specific data and common representation, these objects can be shared among numbers of identical players. Thus loading time and object creating time is greatly reduced.

Achievement: The current game with all required data loaded (20,000 polygons, all color and material parameters, 512 by 512 BMP * 6) takes only about 10MB RAM under Windows XP Pro. This is even smaller than the memory usage of ICQ and Acrobat, and is identical for the sum of opening a WinZip and a Winamp simultaneously.

5.4.4 Fanciness

The problems and their solutions are listed correspondingly.

Problem	*Solution*
Complex terrains are ugly even with large number of polygons.	Use beautiful texture mapping to compensate.
Distant objects are too clear.	Apply fog effects.
The vision of the player is too far.	Implement moving sky box to limit the vision of the player.
Near objects are not detailed enough.	Implement double texturing for near objects (*to be finished*).
Whole terrain takes too much time to render.	Use skybox and view cone to do terrain culling. Also, the speed can be further improved by using rendering detail levels on the terrain object.

Achievement: A virtual scene with realistic look.

5.5 Advanced Topic: Collision Detection

Collision detection is essential for the game. As a module plugged in game logic part, collision detection module is in charge of determining of the following 3 types of events:

- If a fighter is shot by a laser (*object – line*)
- If a fighter is shot by a missile or fighters collide (*object – object*)
- If a fighter has gone into the terrain (*object – object*)

Each of the 3 events must be determined accurately.

The collision detection module is purely software implemented. It supports multi-threading and can be run with multiple instances. It uses a layered architecture with a lot of acceleration methods including

- Bounding box / sphere
- Spatial detection
- Object culling
- Computational optimization

After testing, the module demonstrates satisfactory performance. 3 types of events can be determined with required efficiency. However, the line – object detection is not very accurate.

5.5.1 Features

The collision detection module is a layered module with the following architecture:

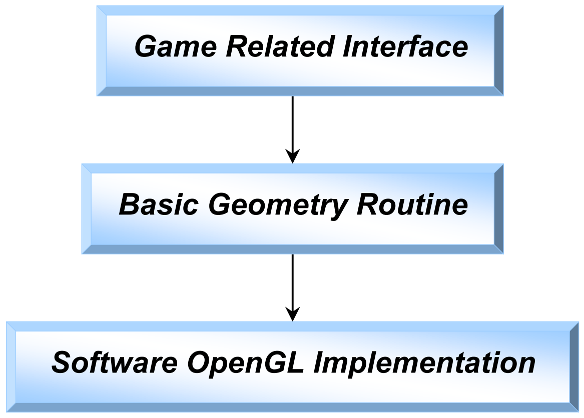


Fig. 5.2 the collision detection module

The main features of the collision detection module include:

- The module fully support object – line, bounding sphere/box – bounding sphere/box and object – height map collision detection.
- The module is purely software implemented and can be run on any computer with any operating system.
- The module supports multi-threading and can be run with several instances.

5.5.2 Problems & Solutions

5.5.2.1 Minimize length of fighter position data from the clients

Problem: The clients have different transformations to the model objects they use. To detect the collision among them, we firstly should re-construct the position of them.

And, since retrieving client information involves network transmission, we want to minimize the amount of data flow on the network. The position information transmission is frequently performed so we want to minimize the length of it.

Solution: There are 2 possible solutions: to transmit the axis vectors, or to transmit the full matrices.

Finally vector transmission is applied. By transmitting vectors we only need to send 9 numbers, while transmitting matrices sends 16 numbers. Although dealing with the vectors at the server side needs more computational power, local computation is always preferable than large amount of data flow, especially for such frequently transmitted data.

5.5.2.2 Re-construct the transformation efficiently

Problem: The transformation matrix should be re-constructed using the provided vectors. The server is assumed to be computationally powerful and we want to utilize the CPU power.

Solution: To fully utilize the computational power of the server, we would like to make the collision detection module to support multi-threading. If the collision module uses OpenGL to compute the matrices, we gain by the hardware computation of the matrices but lose by the AGP interface speed and mutex requirement.

Thus the collision detection module is finally designed as a pure software implementation with required OpenGL operations coded by hand. The transient nature of the module also enables multi-threading supporting. By utilizing the CPU fully and running the module in multi-threading mode, we can maximize the speed of collision detection.

5.5.2.3 Performance fast and accurate collision detection

Problem: Even with all the technique above, the collision detection can still be slow. The problem is the number of triangles needed to be tested is very large. During a single object – object detection there are millions of triangle pairs to be tested, which is definitely a burden to the server.

Solution: The solution to this part is complex and can mainly be divided into 3 categories:

- Bounding box/sphere utilization

  Although large amount of triangle pairs or triangle – line pairs are detected, only very small number of them need the actual accurate detection. Thus we can swap the object by simple geometries like boxes or spheres to filter the unlikely pairs. Since most of the pairs are not colliding, this technique ends up gaining the speed.

  Bounding box/sphere is embedded in model object definition. They can be turned on and off.

- Object culling

  The most time-consuming detection is the terrain – fighter part because of the large number of triangles on the terrain (usually millions of triangles). But again very small proportion of them tends to collide.

  Since the terrain is organized into height map, we can do better than that. Before doing the actual detection, we don’t iterate each of the triangles of the terrain but only do so to a square centered at the position of the fighter with edge length equal to 2 times of the bounding sphere radius of the fighter. Doing this minimizes the triangles to be detected on the terrain.

- Modified collision detection methods

  The logic of the game decides that several modifications can be applied to the collision detection.

  The laser – fighter detection must be fully performed, but bounding box/sphere acceleration can still be applied.

  The missile – fighter detection can be changed into a bounding sphere detection of those 2 objects. This agrees with the actual missile direction method in reality, in which the missile will simple explode when it is within a certain range of the fighter. Same trick can be applied for fighter – fighter detection.

  The terrain – fighter detection can be simplified sine the terrain is a height map. Firstly, we can change the problem into:

  If any of the triangles on the fighter has all the 3 vertices higher than any planes of the terrain, then the fighter doesn't collide with the terrain. If not we need further detection.

  Proof: Firstly, for any triangle to collide the height map, at least 1 vertex of the triangle must be lower than some plane on the height map. If all 3 vertices are higher than the plane, by equation we can get:

  Assume the equation for the terrain is

  $$Ax + By + Cz + D = 0$$

  Thus to say 3 vertices are above the plane it essentially means:

  $$Ax_1 + By_1 + Cz_1 + D > 0$$
  $$Ax_2 + By_2 + Cz_2 + D > 0$$
  $$Ax_3 + By_3 + Cz_3 + D > 0$$

Since any point in the triangle is the linear combination of the 3 vertices, we have for any point P,

$$P = p_1V_1 + p_2V_2 + p_3V_3 \qquad p_1 + p_2 + p_3 = 1$$

Where p1, p2 and p3 are proportions.

Thus we can see that evaluating P in the equation of the plane also get a positive result. In another word, P is also above the plane.

Then for any triangles of the fighter, we can firstly detect if all its pointes are higher than the terrain. If is, we will ignore the triangle. This will speed up the detection process. To further speed up, we can use table to store the already detected points to see if they are above or below the plane. However this can be space-consuming.

If 1 vertex is lower than some plane in the terrain, we will further detect if the intersection line segment overlaps with the plane part of the terrain. This is time-consuming but is only needed for small number of triangle pairs.

5.5.3 Evaluation

Basically, the collision detection module reaches the functional requirement and the performance requirement. But there is some accuracy problem about the line – object detection.

The collision detection module can determine the stated detection required by the game logic, and the performance is acceptable in testing phase.

The accuracy problem is that, sometimes when the laser is pointing to some points near the boundary of the fighter, the result may be wrong. This can be cause by 2 factors:

1. The floating point number calculation has some rounding error.

2. The synchronization of the game makes the vector parameters supplied to the module out-dated.

Generally, factor 1 should not be the major affecting one, while factor 2 is suspected to be. This can only be improved by more flexible messaging mechanism or client-side collision detection.

The 3 layered structure of the collision detection module is fully implemented. The underlying software implementation of OpenGL function is implemented and is proven to be correct, while the basic functions in the 2nd level and the high level cooperation are assumed to contain some computational problems so that the detection against boundary cases, e.g. a very thin and small triangle and a long line far away, may have some problem. In 100 random collision cases tested, more than 95% of the detection results are correct. In 50 cases, the laser – fighter detection is 100% correct, while the terrain – fighter approximation is more than 90% correct in 20 testing cases.

In the current testing process, the collision detection process virtually takes the same amount of time with the synchronization. We have added time counter to monitor the CPU cycle time needed by each thread at the server side, and we get the 3 most time-consuming threads are

1. Synchronization thread
2. Collision detection thread
3. Status broadcasting thread

And the time amount needed by them is virtually **1:2:2** per cycle. Since the synchronization thread is actually running 4 times faster, the collision detection thread is taking less amount of time.

The computational cost for the bounding box/sphere is monitored to be around **0.077** time of the full detection, which proves the usability of this bounding technique.

The terrain – fighter detection, being an approximation, is functioning at more than **90%** of time but will have some situation missed. At the client side there may be some delay equal to the RTT time of the network, but the overall effect is acceptable.

5.5.4 Future Planning

If time permits, 2 improvements can be done to the module:

1. Organize the data in the model into advanced data structure such as DSP tree to better culling process.
2. Utilize more advanced collision detection methods, such as the shortest vector method, and so on.

5.6 Advanced Topic: Terrain Rendering

Following graph illustrated the workflow of the terrain generation process.

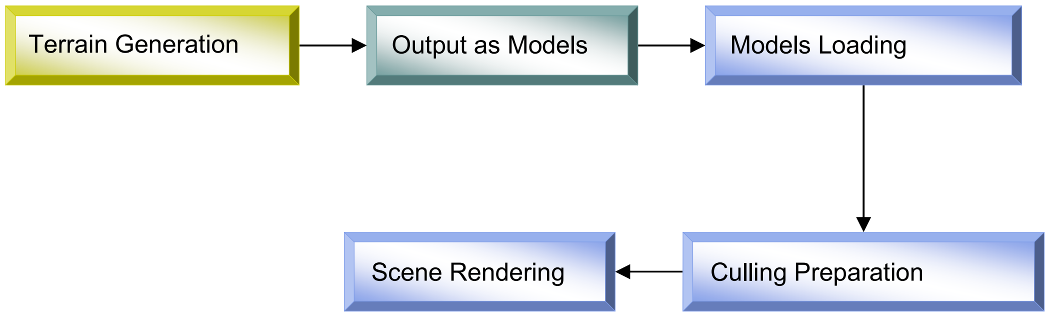


Fig. 5.3 the terrain generation process

The workflow of the terrain rendering process is simple and straightforward. It follows the procedural rendering process of OpenGL architecture, which is consistent with the whole render engine architecture. The detailed description is as follows:

Terrain Generation: The generation of terrains is separated from the core of client side; it is rather like a self-contained program that provides the ability to users to generate their own terrains in an offline mode, independent of the runtime system.

Output as Models: Terrain models would be output as reusable binary model files and XML files. With standardized representation and read/write method, terrain data synchronization can be done easily.

Models Loading: The game logic will load the corresponding file specified by the user and pass the model to render engine.

Culling Preparation: Because of the large number of triangles contained in the terrain, it is desired to only draw the ones visible to the user. Since the terrain is in height map structure, culling technique can be applied to reduce the triangles to be painted.

Scene Rendering: After the culling preparation phase, the terrain will be rendered just as a normal model.

5.6.1 Features

Terrain Generation

Generation of terrain is based on Fractal technique, which makes use of the "self-similarity" property of fractal, plus the introduction of randomness in the generation process.

The algorithm behind the generation is based on "Diamond-Square Algorithm", which is a simplified version of complex random generation process. The idea is to generate the terrain recursively with more and more detailed and refined grids. Here the algorithm is modified to use iteration rather than recursion for speed consideration. The detail of the algorithm is available in the appendix.

The generation is implemented outside the render engine as a separate program. It works as a generation tool that allow user to build their own terrain models in an offline manner, by giving the tool values of core attributes. Moreover the tool also generates texture while it is generating the terrain model, although the textures provide for different model are simple, rooms are provided to user to modify the texture up to their preference. The detail of texture generation is also available in the appendix.

Terrain Culling

Partition, bounding sphere culling and rendering levels are applied to reduce the triangles needed to be drawn as much as possible. Firstly, the terrain is divided into pieces and invisible pieces are removed by detecting against their bounding sphere; then the rendering is carried out with different levels, with more details near the observing point and less further. Doing this the rendering speed and the graphic quality is well balanced.

5.6.2 Problems and Solutions

5.6.2.1 Lack of control over the terrain model with simple model-modelgroup data structure.

Problem: In the first stage of terrain modelling, the data structure used to hold the terrain model parsed by VRML parser is the ordinary model and modelgroup structure, which is dedicated for the data organization of VRML format. This justification is to enable maximum flexibility of rendering. However, this structure is not helpful to terrain culling and partition.

- Partitioning module is affected. Since the only input is the VRML file which holds the terrain information, there is no way to extract essential attributes such as map dimension, map height information, etc. With this information missing, it is difficult to implement partitioning.

- Collision detection module is also affected. Since the collision detection part is originally quite slow, we want to cut the triangle pairs to be detected as much as possible to accelerate. Thus some formatted data is needed, rather than the scattering data defined in normal VRML files.

Solution: Instead of letting users to plug-in their own terrain sources, a terrain generator is provided to them to generate terrain model, with standardized grid representation and height map data structure. By doing this we fix the geometric shape of the terrain as a square grid, and the height map information is also available. This solution solves the 2 problems at the same time, while still maintains the required flexibility for users to have their own terrain.

5.6.2.2 Speed degradation causes by highly complex terrain model

Problem: While it is desirable to achieve high level of flexibility, the efficiency restriction is also present on rendering process of various game objects. Thus the engine is adapted to restrict the model to be formed by only using triangle primitive, plus to extensively make use of all possible

OpenGL optimization technique to handle the uncertainty on the complexity of different models. It works for fighter and missile models, but not for terrain models.

After implementing the height map data structure in order to solve the first problem mentioned above, the generator has the ability to generate terrains of different complexity levels, ranging from 2 polygons to more than 1 millions of polygons. The rendering architecture used before is no longer suitable for terrain scene rendering because of the following 2 major defects:

- Using merely triangles to construct complex terrain will have lots of overheads for such a large model compared to using triangle strip.

- Many resources are wasted in dealing with large amount of polygons that are not seen by the players.

Solution: Instead of using the ordinary rendering strategy, triangle strip primitive is used to reduce the OpenGL overheads. Bounding Sphere Culling method is also implemented to reduce the number of polygons needed to deal with. After that, the terrain is rendered using different levels of details. The most refined scene is rendered near the eye point, while only brief shapes are rendered at far places. Doing all this greatly improve the rendering speed.

5.6.3 Evaluation

The complexity of Diamond-Square algorithm is O(n), which is linear to the number of iterations made. In practice, the iteration takes will around 7 to 10, and the performance of the generation is very efficient with the execution time not more than 2 sec.

The application of terrain culling does shown significant improvement on the rendering performance, however the level of speed up is not as high as the expected level of improvement.

For example, for terrain model of 512 * 512 grids, total 1024 bounding spheres are introduced to partition the terrain into 1024 partitions. In average only 200 – 300 spheres are excluded from supplying to the OpenGL engine for rendering, rather than the original thought of half of the spheres will be pruned in the culling step.

Nevertheless, the introduction of terrain culling in the terrain rendering is proved to be useful. Table 4.1 is a performance comparison on the rendering speed with or without terrain culling for different terrain model complexities.

Number of polygons in the model	Frame rate without culling (fps)	Frame rate with culling (fps)
32768	~ 20	~ 60
131072	~ 10	~ 45
524288	< 5	~ 20

The reason why the performance enhancement is smaller than that expected probably because the terrain landscape change for every small area, the terrain is too detail so that in general the sphere size is larger than necessary, thus cause the culling function to process those grids that are actually not inside the view frustum.

5.6.4 Future Improvements

1. Terrain Generator at this stage can only base on parameters input by users to generate terrain models, users cannot have full control over the appearance and landscape of the generated terrain. Further improvement could be made the generator to allow users to adjust the height values of different points on the terrain, and to provide users with more fancy objects to add to the terrain, e.g. trees, mountains and lakes, and buildings models. A better texture editing tools for building complex texture will also be available.

2. Dynamic texture mapping can be used to simulate a more realistic world. With textures oriented from the same scene but of different resolutions and levels of detail, dynamic texture mapping can be implemented. Depends on the height value of the client fighter position, the higher the fighter the lower the resolution of the texture used to map the terrain, this

thus can stimulate the phenomenon “near object will be clearer than far object”.

3. Better model representation can allow the development of “infinite large terrain”. Although the current data manipulation can create the sense of unlimited terrain scene, the same terrain segment will be repeated infinite times to achieve this, which will be quite fake since every segment is the same. A better representation should allow itself to hold several terrain segments arranged randomly in a single file. This method could improve the appearance of the unlimitedly large terrain which viewing from different direction would give different experience to clients.

5.7 Future Planning

The future planning of the graphics engine mainly focuses on 2 aspects:

- Better the effect of graphics by providing more options in rendering.

  The effect of the graphics can still be improved further. With the current data structure and process, we can create movable parts on the fighters, introduce more particle systems, and provide more refined textures to the objects. The terrain generation process can be further customized to provide more options and controllable parameters for the user.

- Improve the speed by applying more acceleration techniques.

  More advanced culling and collision detection techniques can be applied to improve the speed. Special data structures and algorithms can also be used.

Chapter 6 Object Definition

6.1 Overview

A suitable model can help us to extract useful information from numerous, duplicated and interdependent data, so that analytical process can be done easier in the later steps.

In order to provide flexibility, the system should be able to accept and generate customized models, such as the data and 3D graphics model of fighters and terrains. Hence this part is mainly divided into 3 components: graphics model parser, data model parser and generator.

6.2 Importance of Modeling

Model is a standard for storing certain type of data. A model should contain all necessary information for the object. On the other hand, the information in the model should not duplicate since the size of the model will be huge and the transmission time will also long.

For multimedia data such as graphics, movies or sounds, the compression rate of them is very high, while that of the text data is very low. It is because multimedia data contains much duplicate information and also much of the information is not significant in the view of human being.

By using suitable algorithms and modeling technique, the key information of the data is recorded in a model and the data need to be stored is much smaller in size. Moreover, the key information is usually in text or numerical value, hence further reduce the data size and increase the compression rate. A good model can help us to eliminate such duplication and simplify the information.

6.3 Graphics Model Parser

6.3.1 Why Graphics Model in VRML?

VRML (Virtual Reality Modeling Language) is a language widely used for virtual reality display. It describes a scene so that the user can freely travel in it by means of a viewer. Most of the 3D design and modeling tools, such as 3D Studio MAX, are able to export the 3D graphics in VRML format. The system is able to display the 3D graphics by implementing a VRML parser and accepting the VRML file exported from the 3D design and modeling tools. Also, there are full of 3D graphics model in VRML on the internet. Via VRML, the system is able to display large amount of graphics resource available.

6.3.2 VRML and 3D modeling

VRML can also be used for 3D modeling. In VRML format, the IndexFaceSet Node is responsible for specifying 3D polygons. The texture, color and normals information is stored in the Texture, Color and Normal Node respectively. There is also Transform Node which states the rotation, scaling and translation information of an object. The information is similar to specification of OpenGL. Hence the 3D object in VRML file can be rendered by OpenGL.

6.3.3 Architecture

The graphics model parser is responsible for reading the customized 3D graphics model in VRML format from the users. The parser first reads a file, and checks if it is a valid VRML 2.0/97 file. Then the parser get the necessary data from the file, and then generates an internal Model Class instance which is stored in the centralized data store in the system and transmitted to other parts of the system if needed.

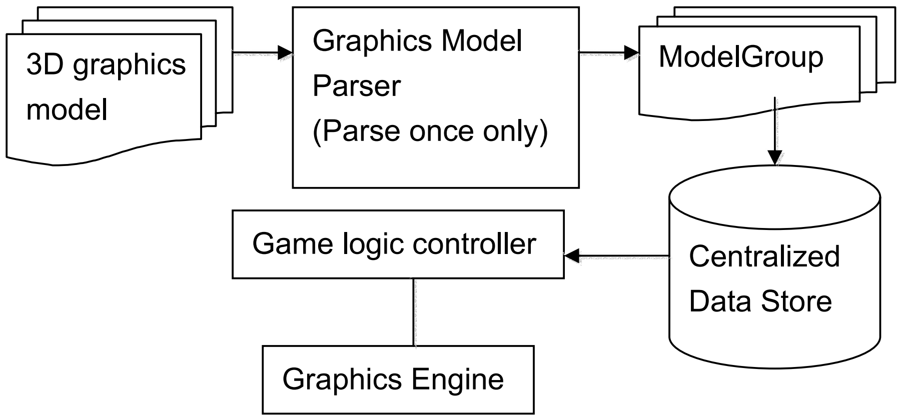


Fig 6.1 the parsing process

Implementing a complete VRML Parser is a sophisticated task, and it is also unnecessary, as the dynamic and audio information in the VRML file is not being handled in the system. As a result, only limited tags and structures is recognized in the VRML Parser and converted into the internal model in the system. Currently, the implementation of the VRML Parser utilizes the CyberVRML library.

As graphics model generator is very complicated, it is currently not included in the system. Users can generate a graphics model in VRML format by using other graphics applications, such as 3D Studio MAX. Then the users can use the model in the system via the VRML Parser.

6.3.4 Problems and Solutions

The following optimization works have been done in order to speed up the parsing time and also the access time of the structure generated.

6.3.4.1 **Array Structure in ModelGroup**

Problem

The parsing task is an off-cycle work, which the VRML file is parsed only once when the system is initialing, and a ModelGroup Class is generated after the parsing work. Then that class is used within the system. Optimizing work should be done on the model to reduce the rendering time in latter stage. Therefore there is a need to optimize the structure in the parsing stage. Although the time of parsing maybe longer, the overall time needed is shorten as the ModelGroup structure is accessed repeatedly.

Solution

The optimizing work mainly bases on the studying of the algorithms used in 3D modeling. The VRML browsers using the OpenGL library usually parse the file and record the structure in recursive way, since VRML is a tree-like structure and OpenGL is a state machine. However, the performance is lowered as many activation records are kept in the memory.

In our project, the data in VRML file is not directly apply to the OpenGL, but being stored into the centralized data store, so that it is better to use iteration rather than recursion to reduce the time for rendering.

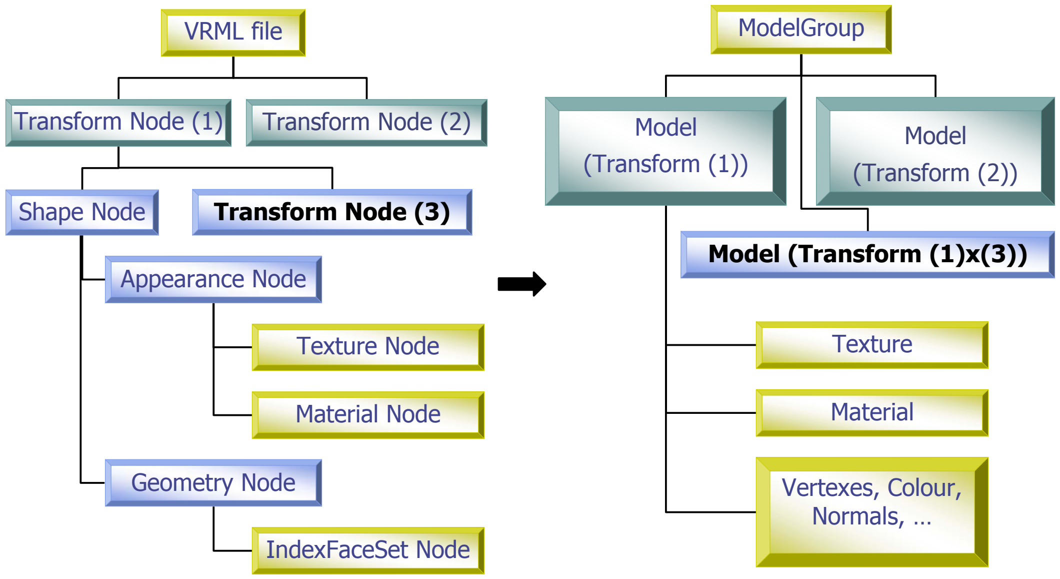


Fig. 6.2 recursive tree structure in VRML is parsed into array of Model Class

When parsing a VRML file, each Transform Node is traversed and all information under this node, including the texture, material, vertexes, colour, normals, is recorded in a Model Class instance. If the Transform Node is a nested one (i.e. there is another Transform Node which is the parent of this Transform Node), the transformation matrix of the Model Class generated will be the multiple of 2 transformation matrixes from the Transform Nodes. Finally a ModelGroup Class, which contains a list of Model Classes, is generated to represent a VRML file. This ModelGroup Class is stored in the centralized data store. In this way, the recursive tree structure of VRML is converted into array of model structure. Therefore, the time accessing the model is reduced and less memory is required. As it is no need to create the complicated nested structure, the parsing time is also reduced.

<u>*Evaluation*</u>

Below shows the time cost for generating the nested structure of model, as well as the list structure of model, for a model in VRML file formed by about 20000 triangles.

The approach	**Time cost**
Nested structure	3 min 40 sec
List structure	3 min 20 sec

6.3.4.2 Limited Support of VRML Tags

<u>*Problem*</u>

In this system, it is no need to have a complete VRML parser, since not all the information in the VRML file is needed. As a result, only limited VRML tags and structures are supported.

However, if too few tags and structures are handled in the system, then the system is not able to display the required 3D model contained in the file, or display in a different form. It will greatly reduce the flexibility of the system. Hence we should strike the balance between the supporting tags and the parsing time.

No matter how many tags and structures are being supported, it is important to note that the parser should be implemented in the way that it does not get run-time error when the VRML file containing the nodes that not being handled in the system.

<u>*Solution*</u>

In VRML, the information belong to the Shape Node is the only visible objects. The background, light source and fog information is ignored since they are already defined in the program. The Transform Node is the parent of Shape Node and it is responsible for storing the transformation needed when displaying the objects specified in the Shape Node. Hence, only the information under Transform Node is identified and retrieved. The other Nodes, such as the Sensor Node, Viewing Node, Environment Node, are simply ignored.

6.3.4.3 Serialized File

<u>*Problem*</u>

VRML file is actually text file. The data in the text file is stored in ASCII characters. It is different from the data in memory which is in binary format. Hence, time is needed to convert the data from ASCII to binary, and vice versa.

<u>*Solution*</u>

The serialization method in CObject is adopted to generate binary format file storing the data. The serialized file is actually memory dump, thus the data in serialized file can be directly copied into memory and there is no conversion time between ASCII and binary format. However, the serialized file is much larger than the original ASCII format text file. Therefore this approach is only used when the data in the file needed to be loaded frequently into the system.

6.3.5 Future Planning

6.3.5.1 Fault Tolerance VRML Parser

Currently, it is assumed that the VRML file imported in the system to have certain constraints. However, a better method is that goes through all the data in the file but ignores the information not satisfy the constraints. One way to do so is to catch the exception during parsing, to avoid the error from causing the failure of whole system.

6.3.5.2 Simplified VRML Parser

Currently, the VRML Parser utilizes the CyberVRML library which parses all labels in the VRML file. It is a time consuming process and it is also unnecessary. A better method is simply ignoring the labels unrelated to graphics display, such as the interactive components.

6.3.5.3 Fighter Graphics Model Editor

There are some constraints for the VRML file for the fighter graphics model, such as the maximum number of polygons, and also the shape of surface, so that it can be rendered in a fast speed. Therefore an editor should be provided to the users so that they can generate the VRML file satisfying the constraints.

6.3.5.4 "Multiple Transformation Nodes" Problem

In this project, the VRML file imported is parsed, and, as stated in the above, each Transform Node is converted into a Model Class. Another way to do this is to multiple the vertexes of the polygons in the model by the transformation matrix. Then the Model Class should have identity transformation matrix (means no transformation is needed) and also, all Transform Node in the same VRML file can be converted into one Model Class. Hence less memory is needed. As parsing is an off-cycle task, the parsing time increased is not significant, comparing to the access time reduced of the Model Class during rendering and transmission. However, it require a fault tolerance parser as many entries of the VRML file can be in vertex base or in surface base, thus it is error-prone to convert them all in one model.

6.4 Data Model Parser and Generator

6.4.1 Why Data Model in XML?

Markup Language

XML (eXtended Markup Language) is a type of markup language.

Markup language uses tags (or markup) to describe the status of the document, such as the font size, the beginning of a paragraph, and also the name of author. Sticky speaking, such information is not part of the document content. However, the information is important for the maintenance and also the processing (especially for the computer applications) of the document.

Markups can be classified into 2 types. Some markups indicate the applications how to process them, such as indicating the browser or printer to display in bold characters. Such markups are called procedural markups as they actually activate a procedure to process that part of the documents.

However, the procedural markups are not flexible. If the appearance of the documents needs to be changed, the whole document is also subjected to the changes. Also, this method is error-prone and provides less portability. For example, if the font specified is not available on a platform, an error is produced.

On the other hand, markups can also be used to describe the structure of the documents. These markups are called general identifiers. General identifiers provided higher flexibility, as we can have different appearance for the documents having same structure. The document can also be processed differently, depends on the platforms; hence we can have higher portability and more customization.

By using the general identifiers to describe a document, the appearance of the document is separate from the content. This, in fact, is helpful to keep the documents tidy and easier for human to read.

SGML, XML and HTML
SGML, XML and HTML are famous examples of markup language.

In fact, they are all simple text documents. The different is that, they utilize tags (<></>) to store the information related to the document. Thus, the documents comprise of characters as well as markups.

HTML enable user to use markups to label the document, and it is widely use in WWW (World Wide Web). SGML is an industrial standard since 1970s, and still being used today. Users are able to define character sets, entities, markups, and also use them. A SGML documents can be dynamically validated against a set of self-defined markups.

However, SGML is complicated, and the developers often spend lots of effort in understanding and developing the self-defined markups. That is one of the reasons why XML is being developed. XML also enables users to define markups, in a simpler ways than SGML. Users can also validate XML documents in run time. As XML is a new standard, it can apply state-of-the-art technology, such as object technology and also the concept of namespace.

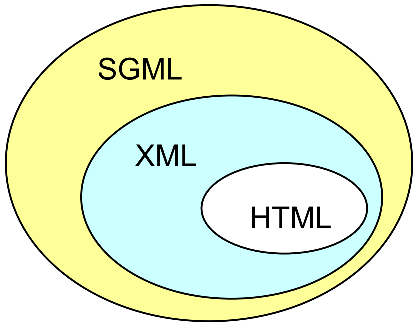


Fig. 6.4 the relationship between SGML, XML and HTML

The relationship between the 3 markup languages is shown in the figure. A valid HTML document is a valid XML document. A valid XML document is also a valid SGML document. Moreover, the later one offers more features then the former.

In general, XML is separated into 3 parts: the structure, content and appearance. The structure defines the meaning of each of the markups as well as the order and occurrence of them. The content part contains the data need to be stored. The appearance part is responsible to translate the data into the format suitable for screen display, and also translate into other type of documents.

SGML	**XML**	**HTML**
Define character set		
Define expressions		
Define entities	Define entities	
Define markups	Define markups	
Use markups	Use markups	Use markups

Fig. 6.5 the abilities of SGML, XML and HTML

The application of XML is mainly divided into 2 categories: visualization for human and processing for programs.

6.4.2 Features of XML

Flexibility

XML provides flexibility, by leaving the right of defining markups to users. The markups are defined in the DTD (Data Type Declaration, for compatibility with SGML) or XML-schema. Hence, XML is a meta language, as it is possible to use XML to define a set of language.

Modularization

An element means a pair of markup. In a model, two or more elements having the same name should not belong to the same level of hierarchy to avoid ambiguity. However, different models may have the same element name in the same level which have completely different meaning. Error may be caused in this way when a document applies more than one model for different purpose (for example, a model is a composition of some other models).

The concept of namespace is applied in XML, to avoid the misidentification of the elements in different models having the same name in a document. Thus, modularization can be done in XML document in order to suit different purpose of applications, to provide higher extensibility.

Human Readable
Different from traditional programming method, XML documents need not be translated or compiled into human non-readable binary code and then processed by computers. The users can directly edit the document as the structure of the document is preserved. There is no need to have compilation before using it. It helps to reduce the debug and hence the development time.

Standard
XML 1.0 (Second Edition) is a W3C (World Wide Web Consortium) recommendation since 2000. Following this standard, there are lots of companion standards provided, such as XKMS (XML Key Management Specification). Once you adopt XML in your application, you are able to utilize the implementations of the companion standards for XML, which costs much less time for development.

Dynamic Validation
It is possible to perform run-time checking for different structures of XML documents, by using a specially designed application called validating parser. The document follows the syntax of XML is called a valid XML document. During validation, the document is validated against the DTD or XML-schema correspondingly. If a valid XML document also fulfills the rules in the DTD or schema, the document is a well-formatted XML document. Validating parsers are often the entry point of XML supporting applications, as the developers need to handle syntax errors themselves.

Portability
When general identifiers are used to describe a document, the document can be processed on many platforms, thus higher portability.

Support from Different Platforms
Being a standard, there are many XML supporting applications available in different platforms, such as validating parser (Xerces, MSXML), XSL Translator (Xalan), and XML signature (IBM XML Security Suite)

Strong Type Language
An XML document should fulfill all the format specifications in the DTD or schema; otherwise the validating parsers should raise error, according to the XML language specification. It helps to reduce the errors in the document and thus the performance of XML-supporting application can be better.

Tree-like Structure
The syntax of XML is defined by a set of EBNF (Extended Backus-Naur Form) grammars of about 80 rules, thus has a tree-like structure. While EBNF cannot be easily understood by human, it is easy for a computer application to validate, parse it and also retrieve the information from the documents.

6.4.3 XML Parsing Method

Different from VRML parsing which needs to utilize custom library, there are common APIs for getting the information in a XML file. They are DOM and SAX. These 2 APIs actually focus on different objectives. There is a comparison in the below to show that which approach is more suitable for this system, and still, efficiency is the main concern.

DOM
DOM (Document Object Model) is an object-based API for parse a document. Each component in XML document is considered as an object in DOM. When a XML document is parsed using DOM, each of the elements is traversed and a tree model is generated, according to the structure of the document. Then, the applications can get the information, by traversing the tree model. Actually, the tree model is same as the XML document in the perspective of the applications.

```
<?xml version=”1.0”?>
<productlist>
<product price=”100.00”>XML Parser</product>
<product price=”150.00”>XML Generator</product>
</productlist>
```
Fig. 6.6 a XML document, pricelist.xml, storing the product information

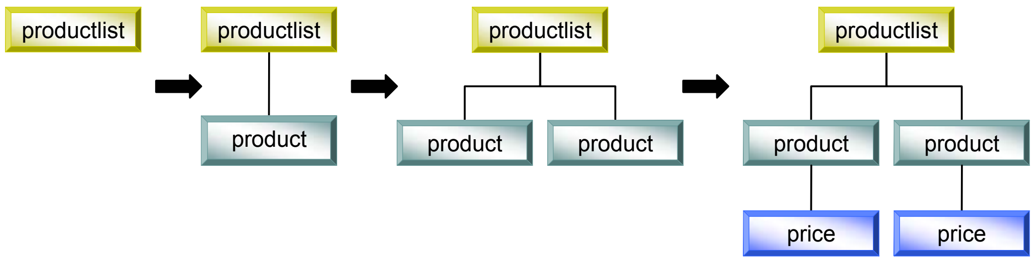


Fig. 6.7 generating the object-based tree model from pricelist.xml

SAX

SAX (Simple API for XML) is an event-based API. The event here does not mean the event from user interface. When a XML document is parsed by SAX, an event is generated when the parser reach the beginning and ending of an element, and even the characters. The parsing job finishes when the parser reaches the ending of the root element. SAX will not preserve the information; this job is the responsibility of the developers.

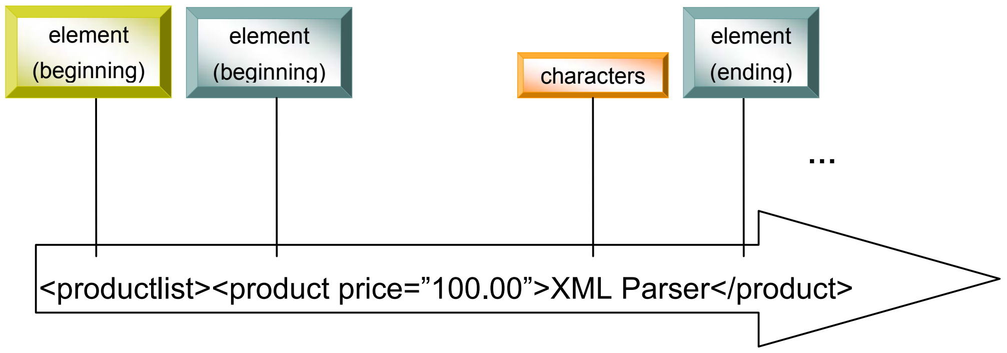


Fig. 6.8 the events generated from pricelist.xml

Comparison between DOM and SAX

Which approach is better?

There is no simple answer for this question. DOM and SAX actually focus on different objectives.

When the developers use SAX, they can have more control on the parsing process. It is also more efficient, as fewer operations are executed and less memory is needed. DOM preserves all document information; therefore it is possible to retrieve the information of the previous part even after the parser has gone to latter part. SAX cannot do it, thus the developers have to record the state of the document.

For many applications, XML processing is only one of the tasks. The major role of the applications is to extract and further analysis the information in the XML document. Most likely there is another data structure for the information, for example, the information is expected to be stored in a database. In this situation, the DOM tree structure is not suitable for the application, and if the application utilizes DOM, it needs to manipulate 2 data structure which actually contain the same information at the same time. There is a duplication problem. SAX is a better solution in this situation, as it does not preserve the information.

DOM can be used in document editing or browsing, since the information of the documents is expected to be read repeatedly but not sequentially. Also, the document updating is provided as a function of DOM.

	DOM	**SAX**
Type of Interface	Object-based	Event-based
Object Model	Created automatically	Must be created by application
Element Sequencing	Preserved	Ignored
Use of Memory	Higher	Lower
Speed of Initial Data Retrieval	Slower	Faster
Stored Information	Better for complex structures	Better for simple structures
Validation	Optional	Optional
Ability to Update XML Document	Yes (in memory)	No

Fig. 6.9 characteristics of DOM and SAX

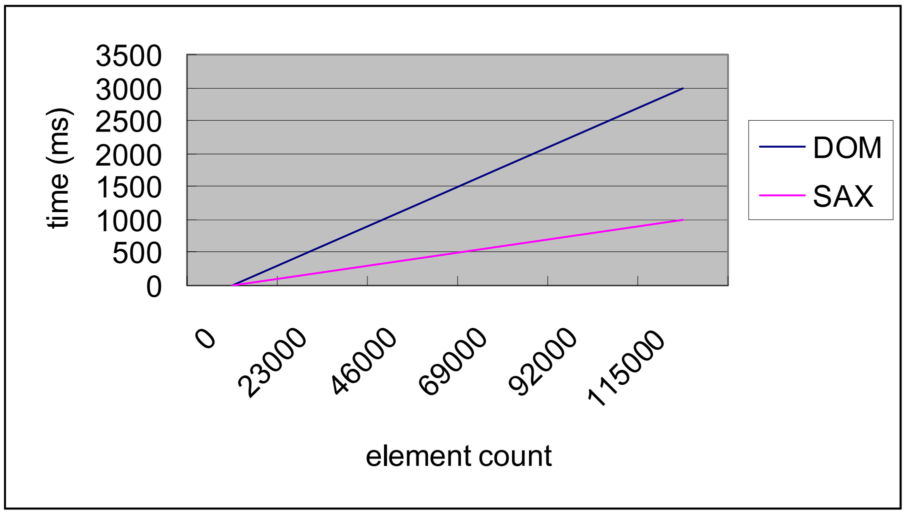


Fig. 6.10 performance of DOM and SAX

Approach adopted in this project

After comparing DOM with SAX, it is significant that DOM will keep all information from the document into the memory, so the memory requirement is higher. SAX does not preserve the information and it is expected that the developers will store the information in some other means. Hence SAX requires less memory.

In the system, the data model parser is used to retrieve the information from the XML file and the model is then copied to the centralized data store. However, we should reduce the load of the system by moving out the parser module after reading the XML document. As DOM will generate a tree model for the XML document, it is expected that the application will continue to retrieve the information from the tree model generated. As a result, the DOM component cannot be moved out since we have to access the model tree using DOM interface. It can be seen that applying DOM does not suit the purpose of our project: optimizing the system to have high performance. Hence, SAX is used in the system to parse the document into the models.

6.4.4 XML generating method

Generate a XML document is just the reverse of parse a XML document. The different is that it is needed to check for error during parsing but not generating. It is because we can control what we write but not what we read.

DOM

DOM can be used in generating XML document. We can edit or generate the tree model of the XML document, and then write the tree model into the file.

SAX

SAX cannot be used in generating XML document.

Custom Method

XML is only simple text document. Therefore the ordinary file writing method can also be used to generate a XML file. This method is much simpler and requires less memory, as it is no need to go through the DOM API. That is the approach taken in this system.

6.4.5 Architecture

The fighter, missile and terrain are expected to be dynamically loaded in the initialization step of the system. As there is no common model standard for them, it is suggested defining models for storing the data. In this system the models are defined in XML. When loading the XML file, the system utilizes MSXML library to parse the file.

Data Model Parser

The XML Parser is able to read XML file in different models, by using different Content Handler. A Content Handler specifies the task need to be done during certain events generated by SAX.

The parser first read the file containing the fighter, missile, or terrain information. Then the information in the file is translated into the model corresponding to it in the system, via the content handler. Finally the model is placed in the centralized data store for further reference of other modules in the systems.

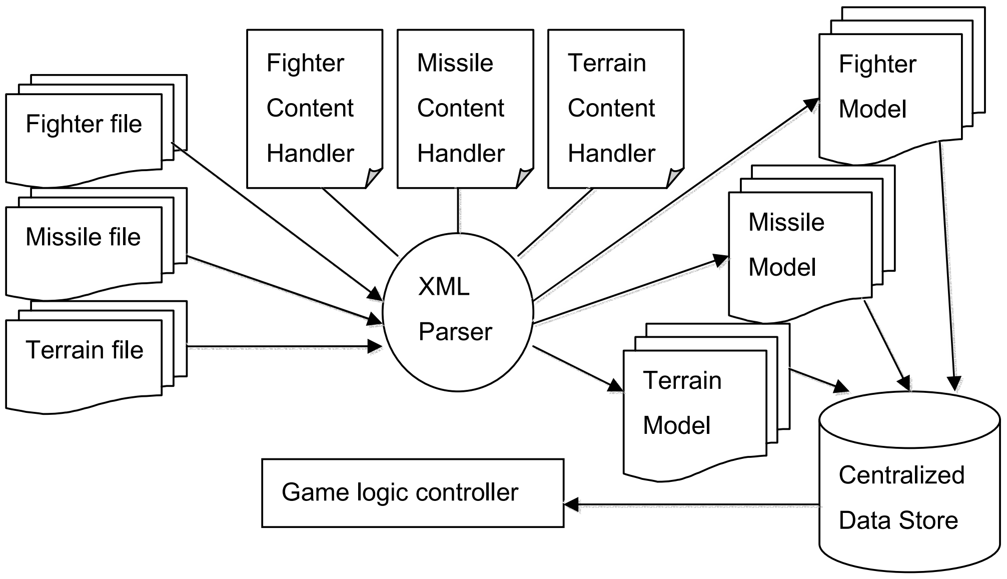


Fig. 6.11 parsing process of XML Parser

Data model Generator
The data model is a usual XML document, so it can be generated by using text editors. However, it is not convenient to do so. The format of XML is well defined. Therefore, it is easy to get syntax error if we generate it by text editor. To provide flexibility, the users should be easy to generate those documents storing the fighter, model or terrain.

As the format of data model is specific for this system, a data model generator should be included in the system, since there is no a solution outside. A user-friendly interface should be provided for the users to generate the models.

6.4.6 Lesson Learned

6.4.6.1 Non-Duplicate Element Names
When parsing a XML document utilizing SAX, each element in the document generates an event. Since SAX will not store the state of the document, it is possible to misidentify the element in different level of hierarchy but have the same name. Besides using a state variable to store which is the current element being parsed, we can also use different name for all the elements to avoid the misidentification problem.

6.4.6.2 Standard not Supported by Library

SAX level 2 is only be implemented in MSXML 4.0, which is not shipped with Windows XP currently. You can only found MSXML 3.0 in Windows XP. However, the dynamic validation function in SAX belongs to SAX level 2.

There are several possible ways to solve this problem.

- Shipping MSXML 4.0 together with the system
- Using MSXML 3.0 without dynamic validation for SAX
- Use dynamic validation for DOM, which is available in both MSXML 3.0 and 4.0. However, using DOM will loss the advantage of using SAX.
- Using other dynamic validating library for SAX implementation

It indicates that a new standard may not easily get supported.

6.4.7 Future Planning

6.4.7.1 Modularization of Model

Currently, the fighter model is expected to have certain constraints, such as having one and only one type of load-out (or machine gun), the type the missile corresponding to certain loading stations. However, real fighters are very complex and having different equipments among the fighters. To be more realistic, the concept of modularization can be applied. For example, a fighter can be further divided into jet engine, wingspan, radar system, loading system, load-out, etc. We can first model these components, and the fighter is the superset of them. We can apply different model to the parts and a fighter comprises of different model of the parts. This can also enhance the reusability of models.

6.4.7.2 Binary XML

GMD-IPSI XQL engine implements PDOM in Java language. The persistent DOM (PDOM) implements the full W3C-DOM API on indexed, binary XML files. Documents are parsed once and stored in binary form, accessible to DOM operations without the overhead of parsing them first. Cache architecture additionally increases performance. It also fully implements the XQL (XML Query Language). It demonstrates that converting a XML document to binary format is feasible.

6.4.7.3 Integrated Model

An Integrated Model is a composition of different type of models. For instance, we can have a single XML file about a fighter, containing the graphics model (X3D), data model, and also a digital signature (XML Digital Signature) to protect the XML file.

- *X3D*
  X3D (Extensible 3D) extends the VRML97 format, which is expected to be the next generation 3D standard. It defines the VRML format in XML syntax. By utilizing XML technology, we can incorporate VRML into XML, and having the advantages of XML stated above.

- Data Model
  The data model stated above, containing the general information of the fighter.

- *XML Digital Signature*
  Digital Signature can be used to prove the source of the model and also protect the copyright of the author. For example, to provide maximum protection, a model having no valid digital signature can be considered as invalid.

When all the models related the same object is stored in a single file, the object is easily to be managed. It is because each XML file stands for a single, independent object, rather than multiple files for an object which the files may depend on each other.

Currently, the model in the files can only be imported from local drive. By using the external entities in XML, it is possible to have Integrated Model which only specifies the URI of the component models.
It is expected that the Integrated Model has a high extensibility, as each of the models can be extended and developed separately. The Integrated Model also has a high flexibility, since the model can be any composition of existing or newly created XML model, by applying the idea of modularization.

However, we should also consider that there may not be implementation about the newly developed XML model at the current time. For instance, X3D is now still in drafting stage.

6.5 Advanced Topic

6.5.1 Extensibility of Data Models

The Data Model is highly extensible when it is defined by XML. By apply XSL (eXtensible Stylesheet Language), we can easily transform XML document into different structure of models.

XSL belongs to the appearance part of XML. It is also defined in XML. One of the typical usages of it is visualizing the document, such as converting XML to HTML for web publishing.

However the functionality of XSL is not restricted to visualization of XML documents. We can extract only certain type of information in the document and generate another document. Because of this property, XSL can be used to extend the model.

Transformation of XML using XSL is completed by using XSLT processor (XSL Transformation processor). One of the famous examples is Xalan of Apache. It accepts a XML document and a XSL document specifying the change in structure, and then generates the XML document in the specified structure.

As mention above, a XML parser can be used to parse different structure of XML document. Content Handlers define the rules of handling different events in SAX. Therefore, we can parse different structure of XML documents by just changing the Content Handler, but not the whole parser implementation.

Hence, we can easily extend our Data Model: by just using XSL to transform between different models and applying different Content Handler for different models.

Chapter 7 Future Trends and Possible Improvements

According to the famous Mall Law of the computer hardware development, the computational resource provided by hardware is expected to increase rapidly. This increase will especially impact the networking part of the game, since the network speed is expected to reach the same level as local storage.

Thus the design concern about network data flow will be lowered and the emphasis will be put more on how to utilize the available computational power of the servers and clients in the best manner. This will probably call for more P2P or distributed applications, which heavily rely on network speed. With improved network bandwidth, the messages can be sent more frequently and the synchronization is not as important as before; while the type of message may change from snap shot of client state into changes in client state.

Graphic process cards will also be improved and is possible to contain more hardware-implemented functions, such as bump mapping, stencil and accumulation buffering, and so on. Thus naturally more features on graphic part are required by players.

Therefore, the quality of the graphics will be improved since more hardware functions are available. However, with the improved graphics quality, the render engine may require several special types of graphics card, which may limit the use of the game on different computers. However, without more sophisticated algorithms, the only solution is resolve to hardware. In addition to this, more advanced algorithms can be implemented to replace the current collision detection and terrain generation parts to make the speed and quality better.

Chapter 8 Conclusion

With the given requirements, people resource and time, the Virtual Reality Fighter gracefully satisfies all the basic requirements and has some additional features, and is thus considered to be acceptable.

As the group members of VRF, we have spent large amount of time and energy on this project. A series of architectures, paradigms, algorithms and models are used, and our skills about design and programming are greatly improved.

Last but not least, we want to give special thanks to the supervisors and tutors of the project. It is because of their help that the project can be finished with required functions and within given time.

Appendix

A1. Pseudo algorithm for terrain generation

Diamond-Square Algorithm

The algorithm is basically a mid-point displacement algorithm that is widely used in fractal generation. The algorithm mainly consists of 2 important steps, the Diamond Step and the Square Step.

The diamond step: Taking a square of four points, generate a random value at the square midpoint, where the two diagonals meet. The midpoint value is calculated by averaging the four corner values, plus a random amount. When multiple squares are arranged in a gird diamonds are formed.

The square step: Taking each diamond of four points, generate a random value at the center of the diamond. Calculate the midpoint value by averaging the corner values, plus a random amount generated in the same range as used for the diamond step. This will form squares again.

The data structure used to implement the algorithm is a 1D array which is stimulating a 2D array structure. At the initial setup, as no data is in the array and therefore there is no way for the algorithm to proceed. To take care of this, in practice, some values will be "seeded" into the array at the position of the 4 corners, to act as initial data for the first pass diamond step.

Fig 4.1 shows the result of the first pass of the 2 steps:

a. The array initially seeded 4 values in the corners, in practice those 4 values will be the same.
b. This is the first diamond step. The values in the 4 corners are averaged, plus a random value to give a new value to the center of the square.
c. This is the first square step. At this stage, the four diamonds formed in the previous step are wrapped. Taking the 4 corners' values of each diamond and the center value is calculated.

Figure d and e shows the 2 steps in the second pass.

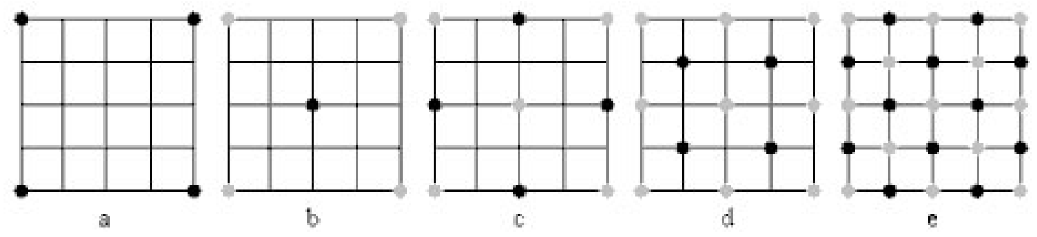


Fig 4.1 Simple example showing the effect of the first 2 passes of the Diamond-Square algorithm.

Notice that the first pass of running the algorithm yields 4 squares, the second pass would end up with 16 squares. The number of squares generated is equal to $(2^I)^2$, where I is the number of iterations through the recursive subdivision routine.

Number of iteration controls how detailed will the resulted terrain be. Fig 4.2 and 4.3 illustrate the difference between terrain models of 2 commonly used number of iteration values in our game.

Besides number of iteration, the random seed value is also a very important attribute to control how the terrain model is formed. Fig 4.4 and 4.5 demonstrate the difference between terrain models of 2 random seed values.

Table 4.1 shows the information of the 4 sample terrain models.

	Figure 4.2	**Figure 4.3**	**Figure 4.4**	**Figure 4.5**
Number of Iterations	8	7	9	9
Number of Grids	65536	16384	262144	262144
Number of Triangles	131072	32768	524288	524288
Random seed value	0	0	9999	0

Table 4.1 Information of terrain model Fig. 4.2, 4.3, 4.4 and 4.5

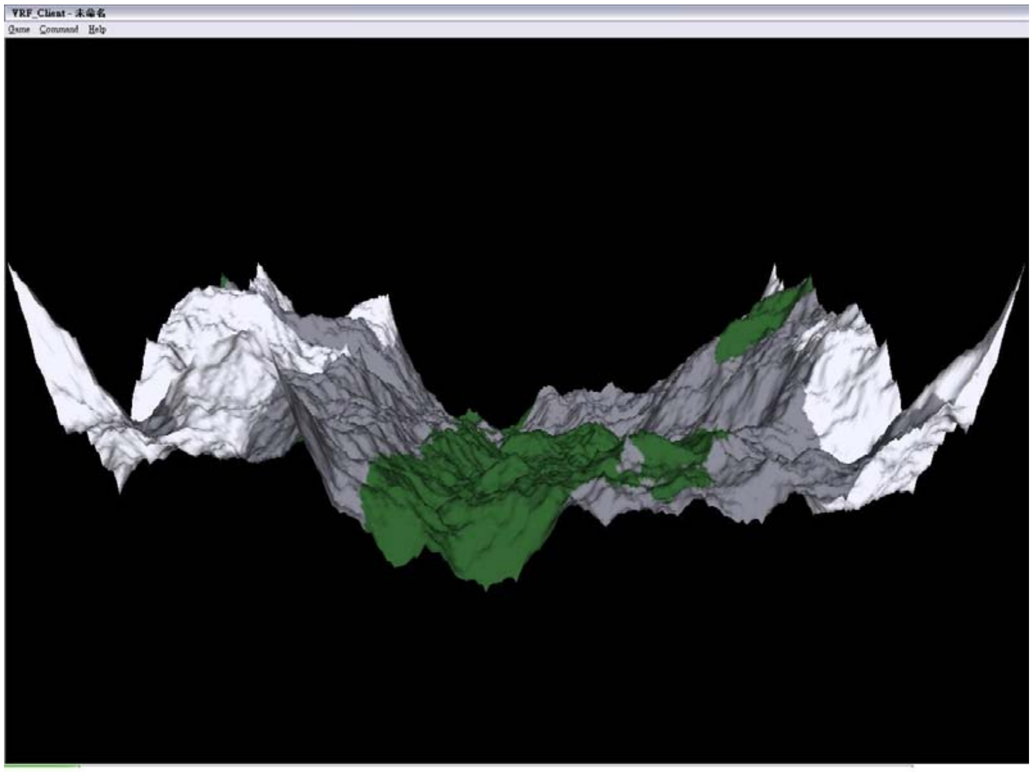


Fig 4.2 Terrain model with l = 8 and random seed = 0

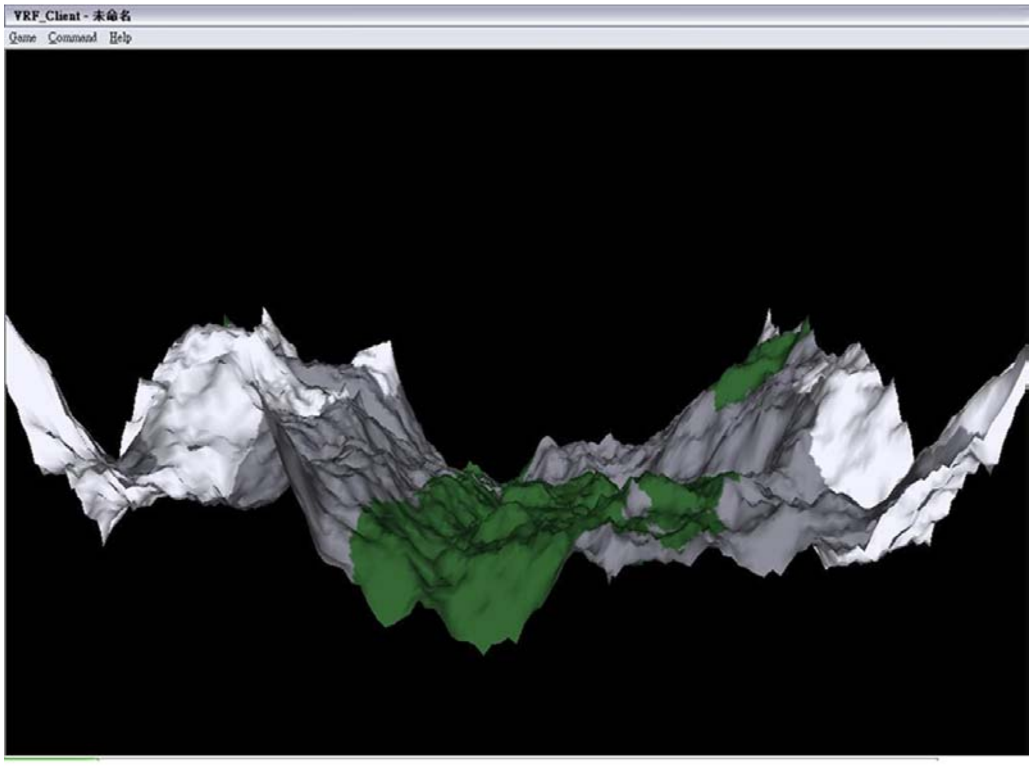


Fig 4.3 Terrain model with l = 7 and random seed = 0

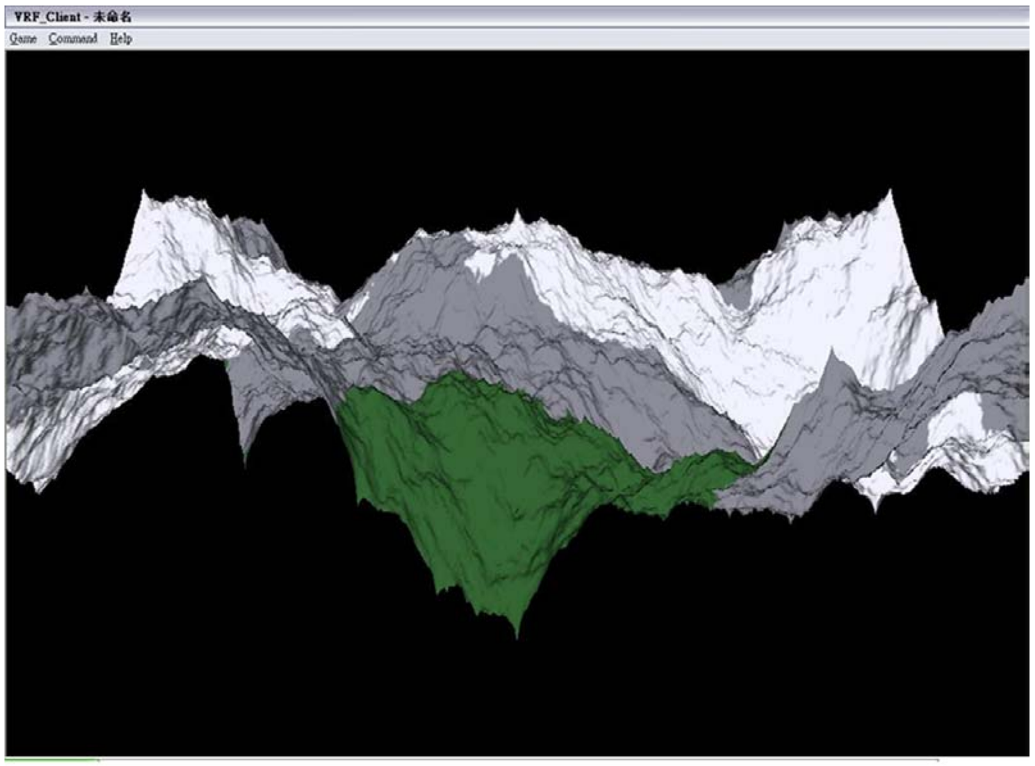


Fig 4.4 Terrain model with I = 9 and random seed = 9999

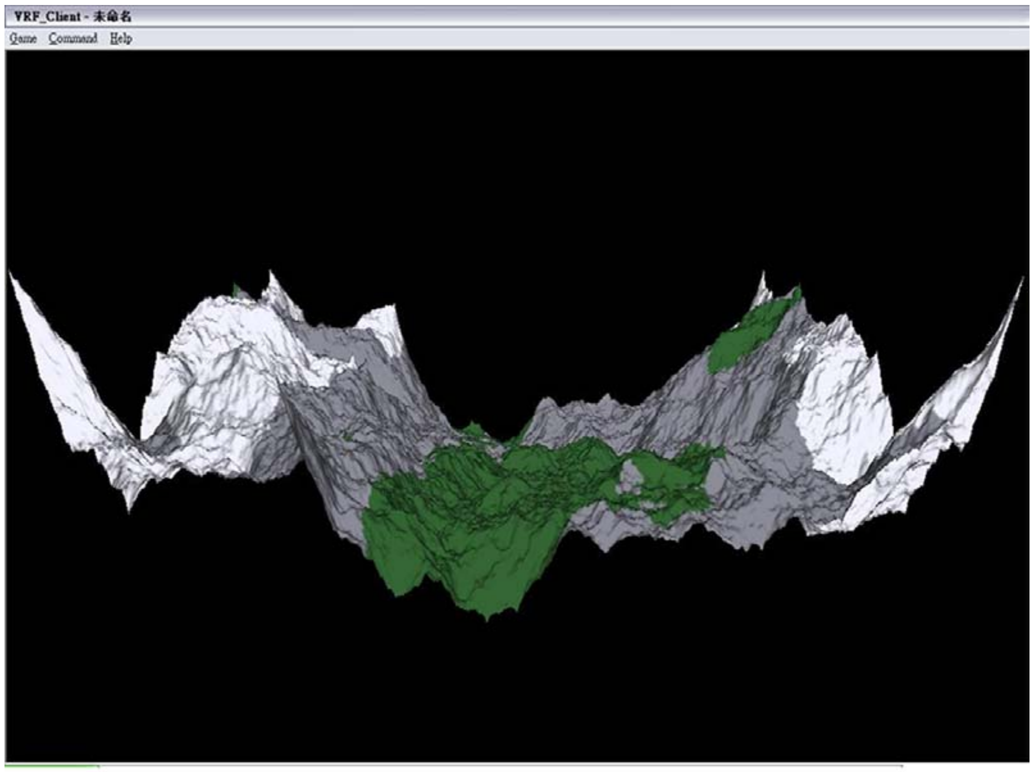


Fig 4.5 Terrain model with I = 9 and random seed = 0

A2. Texture Generation

With the height map information resulted by running the Diamond-Square algorithm, the texture generation is straightforward. In the implementation, the function tries to group the height values into 3 groups, then artificially assigning 3 different colors to those 3 groups. In the example, the higher group is assigned to white, the middle group is assigned to grey and the lower group is assigned to green. Fig 4.6 and 4.7 shows 2 different textures generated by using height map resulted from using random seed values 0 and 9999.

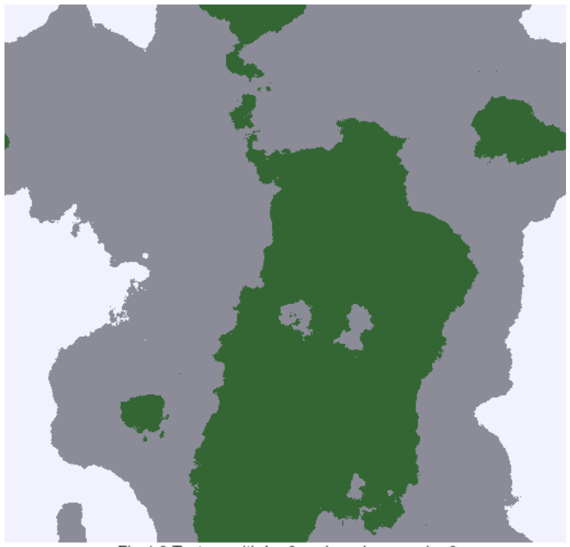


Fig 4.6 Texture with l = 9 and random seed = 0

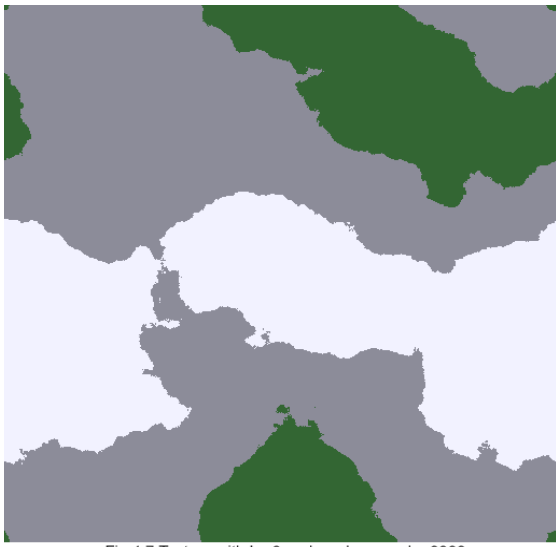


Fig 4.7 Texture with l = 9 and random seed = 9999

At this stage textures created by the tool are very simple color blends, but indeed provides a preliminary coloring for the terrain model. User can modify the texture up to their preference and this simple texture can act as “visual” height information.

A3 Terrain Culling Technique

Terrain culling is done by applying Bounding Sphere – View Frustum detection mechanism. The basic idea is to partition the grid map into a number of larger grids, in practice size of 16 x 16, 8 x 8 or 4 x 4 grid sizes are used, and with each of the larger grid surrounded by a bounding sphere. With the four normal vectors of the view frustum are ready, it is very easy to determine whether a particular sphere is inside or outside the view frustum. Fig 4.8 shows the general idea of the detection:

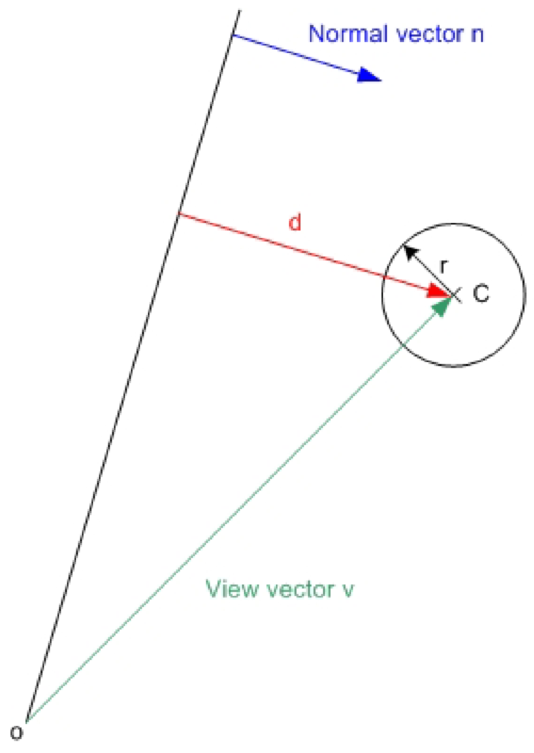


Fig 4.8 Diagram shows the general idea of terrain culling detection

If d < r then the sphere is inside the view frustum.

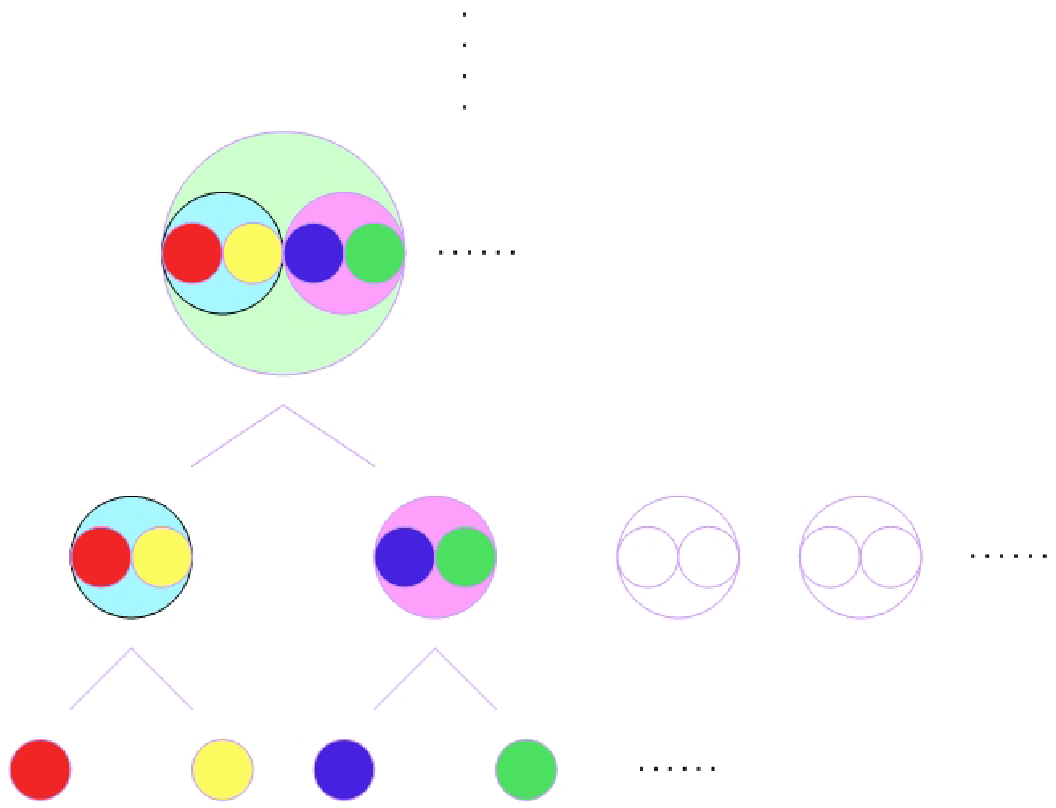
If d = r then the sphere is touching the outer side of the view frustum.

If d > r then the sphere is completely outside the view frustum.

And we can obtain d by v · n. Hence if v · n < r the sphere is inside the view frustum and the grid bounded by that sphere should be rendered, otherwise the grid is not considered.

This method indeed reduces the number of polygons need to be considered by the engine, in general the number of spheres used to bound the whole terrain are 1024 – 4096. With this huge amount of spheres to be checked in every frame it is still not very efficient.

One method to use is binary search tree. Fig 4.9 shows how the tree is built. Basically instead of checking the spheres directly, we recursively group 2 adjacent spheres together and add a new and bigger bounding sphere to bound this sphere group. By doing so the searching speed will be reduced from O(n) to O(lg(n)).



A4 Universal Message Scheme

Critical Message

Data	SERVER_TIME_REQ	SERVER_TIME_ACK	PLAYER_ID_REQ	PLAYER_ID_ACK
1	Sequence No	Sequence No	Sequence No	Sequence No
2	Message Type	Message Type	Message Type	Message Type
3				Fighter ID
4			Fighter Type	
5				
6				
7				Fighter Position(x)
8				Fighter Position(y)
9				Fighter Position(z)
10				Fighter Head (x)
11				Fighter Head (y)
12				Fighter Head (z)
13				Fighter Up (x)
14				Fighter Up (y)
15				Fighter Up (z)
16				
17				
18				
19	TimeStamp (second)	TimeStamp (second)	TimeStamp (second)	TimeStamp (second)
20	TimeStamp (millisecond)	TimeStamp (millisecond)	TimeStamp (millisecond)	TimeStamp (millisecond)

Data	SYNC_DATA_REQ	SYNC_DATA_ACK	START_GAME_MSG	FIRE_MISSILE_REQ
1	Sequence No	Sequence No	Sequence No	Sequence No
2	Message Type	Message Type	Message Type	Message Type
3	Fighter ID	Fighter ID	Fighter ID	Missile ID
4		Fighter Type		Missile Type
5				Target Fighter ID
6				Partition ID
7		Object Position(x)		Missile Position(x)
8		Object Position(y)		Missile Position(y)
9		Object Position(z)		Missile Position(z)
10		Object Head (x)		Missile Up(x)
11		Object Head (y)		Missile Up(y)
12		Object Head (z)		Missile Up(z)
13		Object Up (x)		Missile Head(x)
14		Object Up (y)		Missile Head(y)

15		Object Up (z)		Missile Head(z)
16		Object Type		
17				
18				
19	TimeStamp (second)	TimeStamp (second)	TimeStamp (second)	TimeStamp (second)
20	TimeStamp (millisecond)	TimeStamp (millisecond)	TimeStamp (millisecond)	TimeStamp (millisecond)

Data	FIRE_LASER_REQ	CLIENT_EXIT_REQ	CLIENT_EXIT_ACK
1	Sequence No	Sequence No	Sequence No
2	Message Type	Message Type	Message Type
3	Fighter ID	Fighter ID	Fighter ID
4		Exit Code	
5			
6			
7	Laser Position(x)		
8	Laser Position(y)		
9	Laser Position(z)		
10	Laser Up(x)		
11	Laser Up(y)		
12	Laser Up(z)		
13	Laser Head(x)		
14	Laser Head(y)		
15	Laser Head(z)		
16			
17			
18			
19	TimeStamp (second)	TimeStamp (second)	TimeStamp (second)
20	TimeStamp (millisecond)	TimeStamp (millisecond)	TimeStamp (millisecond)

Data	CHANGE_PARTITION_MSG (Inter-server)	CHANGE_PARTITION_MSG (Clients)	GAME_END_MSG
1	Sequence No	Sequence No	Sequence No
2	Message Type	Message Type	Message Type
3	Fighter ID	Fighter ID	
4	Fighter Type	New Server ID	
5	Player Name		
6			
7	Fighter Position(x)		
8	Fighter Position(y)		
9	Fighter Position(z)		
10	Fighter Up (x)		
11	Fighter Up (y)		
12	Fighter Up (z)		
13	Fighter Head (x)		
14	Fighter Head (y)		
15	Fighter Head (z)		
16	Fighter HP		

17	Fighter Fuel		
18	Missile Remain		
19	TimeStamp (second)	TimeStamp (second)	TimeStamp (second)
20	TimeStamp (millisecond)	TimeStamp (millisecond)	TimeStamp (millisecond)

Non-Critical Message

Data	UPDATE_FIGHTER_STATUS	UPDATE_MISSILE_STATUS
1	Sequence No	Sequence No
2	Message Type	Message Type
3	Fighter ID	Missile ID
4	Partition ID	Partition ID
5	HP	Target Fighter ID
6	Fuel	
7	Fighter Position(x)	Missile Position(x)
8	Fighter Position(y)	Missile Position(y)
9	Fighter Position(z)	Missile Position(z)
10	Fighter Up (x)	Missile Up(x)
11	Fighter Up (y)	Missile Up(y)
12	Fighter Up (z)	Missile Up(z)
13	Fighter Head (x)	Missile Head(x)
14	Fighter Head (y)	Missile Head(y)
15	Fighter Head (z)	Missile Head(z)
16		
17		
18		
19	TimeStamp (second)	TimeStamp (second)
20	TimeStamp (millisecond)	TimeStamp (millisecond)

References

Web Site

An introduction to octree
http://www.flipcode.com/tutorials/tut_octrees.htm

Apache XML Project (Xerces, Xalan)
http://xml.apache.org

CyberVRML
http://www.cybergarage.org

Document Object Model (DOM)
http://www.w3.org/DOM

Extensible Markup Language (XML) 1.0 (Second Edition)
http://www.w3.org/TR/REC-xml

GMD-IPSI XQL Engine
http://xml.darmstadt.gmd.de/xql

Internet-Based Virtual Environments
http://vehand.engr.ucf.edu/handbook/Chapters/Chapter19/Chapter19.html

Load Balancing Schemes for Distributed Real-Time Interactive Virtual World Simulations
http://etd.uwaterloo.ca/etd/ijcunnin2000.pdf

Management of Networked Virtual Environments
http://cs.ulb.ac.be/publications/MT-02-01.pdf

MSXML
http://msdn.microsoft.com/xml

Octree tutorial
http://www.gametutorials.com/Tutorials/OpenGL/Octree.htm

Proksim Software Inc. (Multiplayer real time game architecture)
http://www.gdconf.com/archives/proceedings/2000/Proksim.doc

Simple API for XML (SAX)
http://www.saxproject.org

Virtual Reality and Model Building
http://www.cv.iit.nrc.ca/~cs410/downloads/netves.ppt

VRML97 Specification
http://www.web3d.org/Specifications/VRML97

Web 3D Consortium
http://www.web3d.org

World Wide Web Consortium (W3C)
http://www.w3.org

X3D
http://www.web3d.org/x3d

Papers

http://www.gamers.org/dEngine/quake/QDP/qnp.html

http://www.gamers.org/dEngine/quake/info/techinfo.091

http://www.gamasutra.com/features/19990903/lincroft_01.htm

http://www.gamasutra.com/features/19970801/ng.htm

A Distributed Multiplayer Game Server System

Eric Cronin Burton Filstrup Anthony Kurc
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109-2122
ecronin,bfilstru,tkurc@eecs.umich.edu
May 4, 2001

Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments

Thomas A. Funkhouser and Carlo H. S´ equin
University of California at Berkeley