

Give and Take 3D Network Game(GNT) ver 0.8

Jae-Hyuck Yoo, Jae-Hyun Kim

Feb. 2003

Table of Contents.

1. Introduction.
2. Basic game program structure.
3. All of GNT.
4. Endnotes.

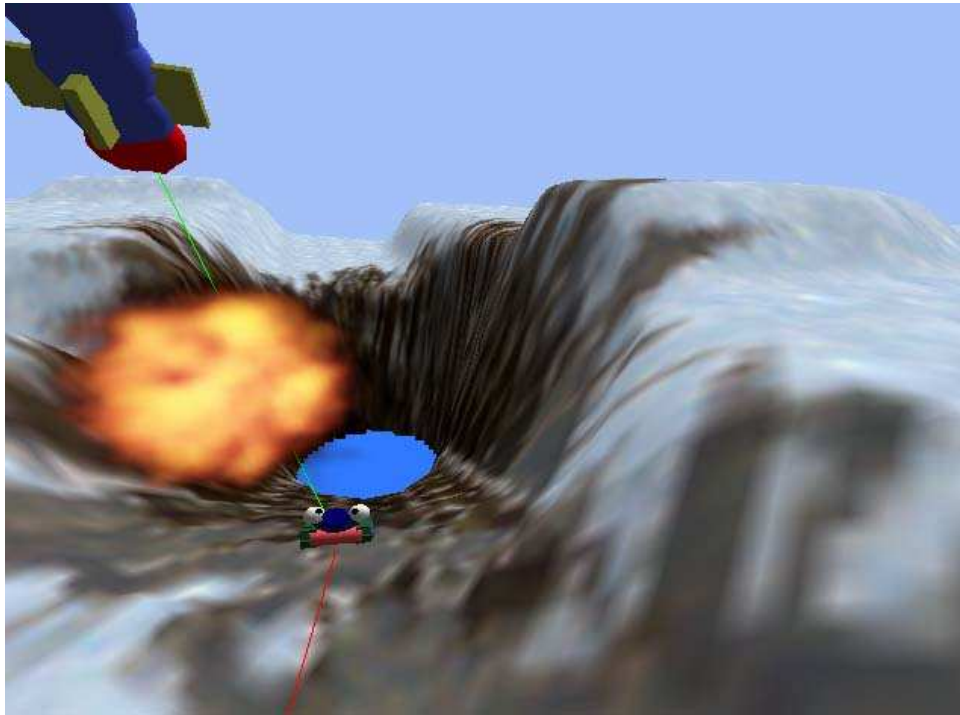


Figure 1: screen shot.

1 Introduction

1.1 Reason of this document birth.

GNT Project is for development of game programming skill of DANAL RND. So, the acquired skill have to be shared with DANAL RND members at least by this document.

1.2 Features of GNT.

1. Three dimension(3D).
2. Use of server and client(C/S) and peer to peer(P2P).
3. Turn system.
4. Quad tree for 3D map effects.
5. Use of direct 3D(9.0) series.
6. 3D sound.
7. Fast file config [FFS]

2 Basic game program structure.

Flow of game is divided into three parts. First, input. Second, processing. Third, output. They can run simultaneously.

2.1 Input part

Game is interactive. If a game ignores commands of player, the player would be angry so much. To capture all of commands of player, almost games use **Direct Input**. Moreover the commands can be sent to other peer or server through network. So, command buffer is needed not to miss them. **Direct Play** is also nice to make P2P network. Conclusion, almost network games use **Direct Input** and **Direct Play** to get commands, and it is important to preserve the commands before use. GNT use also **Direct Input** and **Direct Play**. Following is simple introduce of **Direct input and play**. Important part is emphasized by **bold font**.

2.1.1 Direct Input

Apart from providing services for devices not supported by the Microsoft Win32 API, **DirectInput gives faster access to input data by communicating directly with the hardware drivers rather than relying on Microsoft Windows messages.**

DirectInput enables an application to retrieve data from input devices even when the application is in the background. It also provides full support for any type of input device, as well as for force feedback.

Through action mapping, applications can retrieve input data without needing to know what kind of device is being used to generate it.

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

2.1.2 Direct Play

The Microsoft DirectPlay application programming interface (API) is the component of Microsoft DirectX that enables you to write network applications such as multiplayer games. DirectPlay performs all of the hard work associated with connecting players, even those behind **Network Address Translation (NAT)** devices, and managing sessions. It allows you to create, find, and connect to multiplayer games. Once connected, DirectPlay

enables you to send guaranteed or non-guaranteed messages to other players. A common framework for launching applications and in-game voice communications is also provided.

DirectPlay provides a layer that largely isolates your application from the underlying network. For most purposes, your application can just use the DirectPlay API, and enable DirectPlay to handle the details of network communication. DirectPlay provides many features that simplify the process of implementing many aspects of a multiplayer application, including:

- Creating and managing both **peer-to-peer** and **client/server** sessions
- Managing users and groups within a session
- Managing messaging between the members of a session over different network links and varying network conditions
- Enabling applications to interact with lobbies
- Enabling users to communicate with each other by voice

All of upper aspects are available through same interface by easy options.

2.2 Processing part

If catching commands part is completed, core part of game is left. Calculating and transformation by command is needed. Almost game program architecture made up of 0. Receive Command 1. Frame Move 2. Render. **Input Part** is responsible for 0. Receive Command (plz, refer to upper).

2.2.1 FrameMove

There are must datas for game progress. Caculate, transform and update the datas along with the commands. For example, if there are commands to 'go' and 'left', programmer have to caculte the direction vector, normal vector by multifying velocity of the object, and update the position vector of it. Of course, more operation is needed.

It is certain that FrameMove is responsible for correct data of object by commands.

2.2.2 Render

For **DirectX Graphics**, called Direct3D, draw something in screen, you must know the internal principal to draw. There are two surfaces, and you can access just one surface. The **DirectX Graphics** flip between them. So, you have to draw all object on the surface from datas which update **FrameMove**.

You can use vertex or mesh to present an object. Vertex is the structure that have all of information of a point. It must have 3D point, can have color, image info, light info, ... Mesh is a complex vertex collective file plus more info, **DirectX Graphics** has 'X' file format to present mesh.

It is important that all object to draw must save in the surface to be flipped.

2.3 Output part

There are two regions human can feel from current common pc. First, sight. Second, sound. **DirectX Graphics** does all about visual effect, **Direct Music** dose all about sound effect. (there are also **Direct Sound** but i didn't use it)

2.3.1 DirecX Graphics

This is an illustration of the graphics pipeline. The functionality in each of the blocks is introduced below and contains links to more information. Below is very important concept.

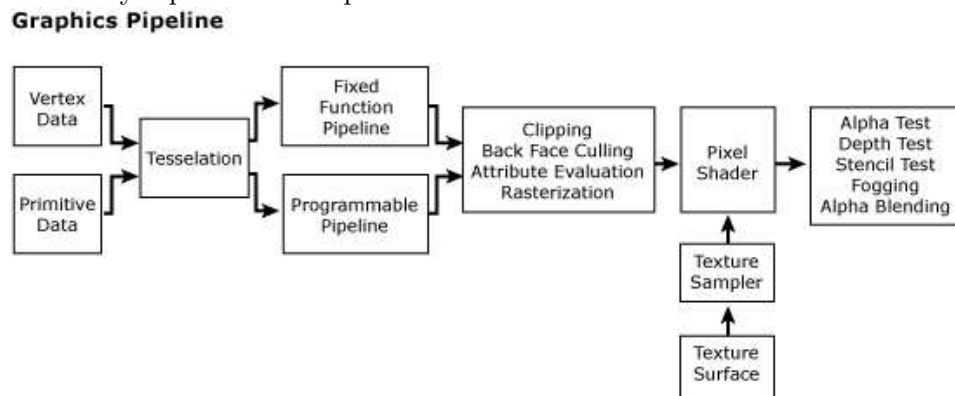


Figure 2: the graphics pipeline.

The following diagram shows the relationships between Microsoft Direct3D, the Microsoft Windows Graphics Device Interface (GDI), the hardware abstraction layer (HAL), and the hardware.

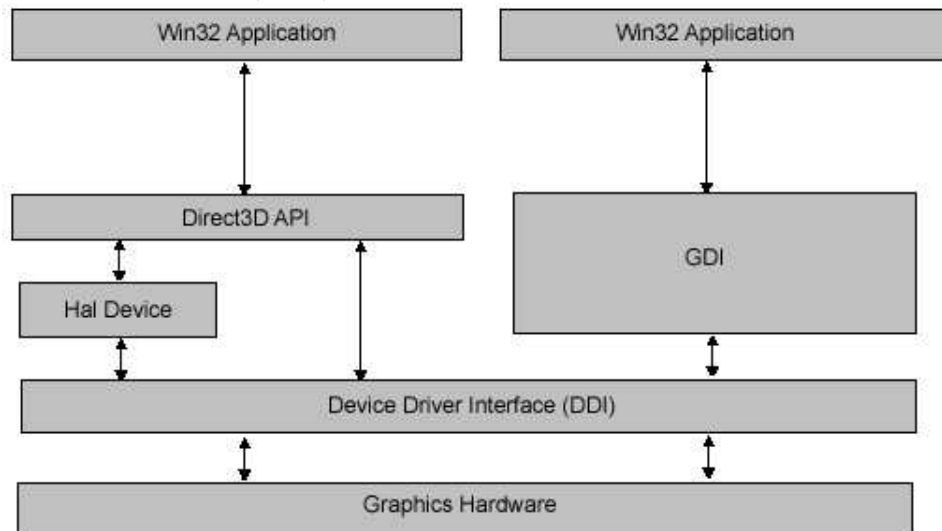


Figure 3: System Integration.

As the preceding diagram shows, Direct3D applications exist alongside GDI applications and both have access to the graphics hardware through the device driver for the graphics card. Unlike GDI, Direct3D can take advantage of hardware features when a HAL device is selected. HAL devices provide hardware acceleration based on the feature set supported by the graphics card. You are provided with a Direct3D method to determine at run time if a device is capable of the task.

2.3.2 Direct Music

By using the DirectMusic interfaces in your application, you can do the following:

- Load and play sounds from files or resources in MIDI, WAV, or DirectMusic Producer run-time format.
- Play from multiple sources simultaneously.
- Schedule the timing of musical events with high precision.

- Send tempo changes, patch changes, and other MIDI events programmatically.
- Use Downloadable Sounds. By using DLS, an application can be sure that message-based music sounds the same on all computers. An application can also play an unlimited variety of instruments and even produce unique sounds for individual notes and velocities.
- Locate sounds in a 3-D environment.
- Easily apply pitch changes, reverberation, and other effects.
- Use more than 16 MIDI channels. DirectMusic makes it possible for any number of voices to be played simultaneously, up to the limits of the synthesizer.
- Play segments on different audiopaths, so that effects or spatialization can be applied individually to each sound.
- Capture MIDI data, or stream ("thru") it from one port to another.

2.4 Others

It is important to synchronize sequence in network game. You can learn all of detail programming method very kindly from tutorial. If you want more help about game programming then, mail to febace@febace.com

3 All of GNT.

3.1 GNT framework.

Next is detail list of GNT implemenation.

1. Phase driven play.
2. Manager class.
3. Factory class.
4. GUI control classes.
5. Quad tree for 3D map effects.

3.1.1 Phase driven play.

GNT have six phases. Always there is just one running phase. Previous phase is responsible for pass the right to run to next phase by itself.

1. MainMenu phase
 - connect to C/S server.
 - login authentication.
 - virtual session created.
2. WaitRoom phase
 - create or join game room.
 - chatting with other user.
3. GameRoom phase
 - change tank or team.
 - chitting with other user.
 - ready or start game.
4. Preparation phase
 - initialise the game.
 - P2P network are launched.
 - Map, tanks are created.

5. GamePlay phase

- process command
- frame move, render
- real GNT game is here.

6. Result phase

- score of the game before.

3.1.2 Manager class.

There are four managers to manage all of related with it. You cannot access to object directly, but by this class. Game play phase orders not to tank each, but to all manager, then manager orders to tank.

Single-tone pattern is used. Single-tone pattern is useful, when there have to just one instance. It is like a global data.

1. CommandManger.

- manage all commands from P2P, local(keyboard, mouse), C/S.
- insert them to queue - enqueue.
- pop them from queue - dequeue.

2. ControlPanelManager.

- manage all of control panel objects.
- minimap, power bar, energy bar, move bar.
- angle, direction browser.
- zoom bar.

3. InfoPanelManager - not implemented.

4. PlayGroundManager.

- manage all of object in playground.
- map, tank, tree, building.
- core part of GNT.

3.1.3 Factory class.

Factory class is responsible for making, deleting, storing of objects. There are object lists and pools, object list is useful to search and update. object pool is useful to speed up. Because making some object in game running need a few time.

Factory pattern is used.

3.1.4 GUI control classes.

There are button, dialog, editbox, listbox, message box, scrol bar, static. All of these control are made newly.

3.1.5 Quad tree for 3D map effects.

First time make the map by just vertex buffer, speed of render is terrible. After looking up again a few algorithms for effiecient vertex managing, first is binary tree. But its result not enough fast to game run. Second is Quad tree method, its result is enough to play game. :)

3.1.6 Deatail of quad tree

The **L.O.D** is abbreviation of "Level of detail". In summary, **LOD** is the way to draw detailedly if near and inversely. **GNT LOD** use the quad tree for map vertex info buffering. The quad tree update by camera position changed or missile explosion. Detail of level make by shape of square. All quad tree leaf filled by caculation error check routine, with map vertex info when **FrameMove**. And all quad tree's vertex info filled before insert into the D3D surface when **Render**.

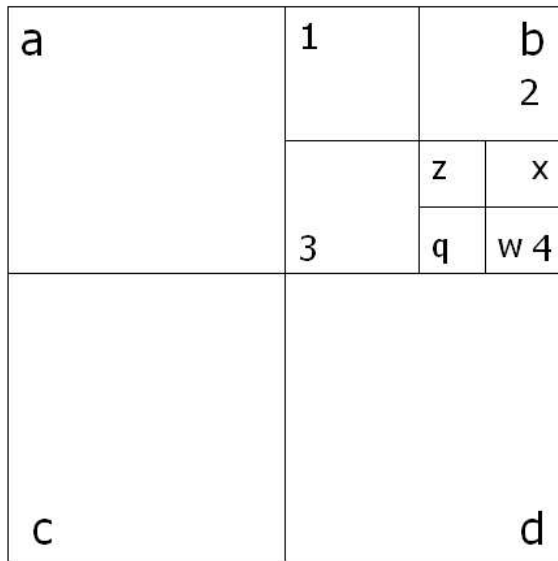


Figure 4: diagram to show vertex info in quad tree.
The way of LOD is like upper figure.

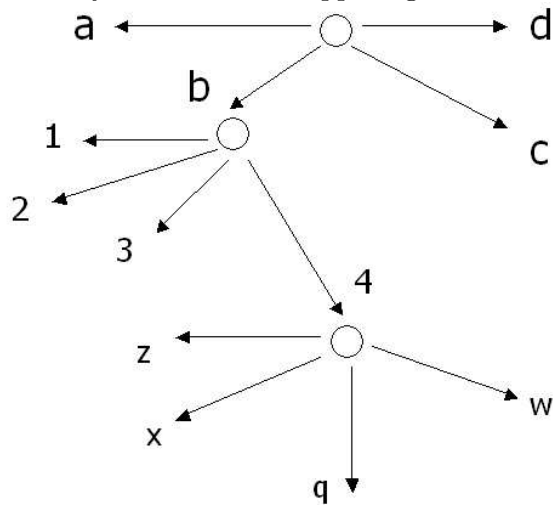


Figure 5: quad tree of the upper diagram.

More quad tree added, when error bigger by camera moving or missile explosion. The error check routine think of camera position, error between map raw file value, etc... If error is upper the error limit, quad leaf is added to the tree.

pf. GNT quad tree use 'reference parameter' not to use pointer. Quad

tree has no parent pointer, but receive reference parameter when called. So more faster than dynamic memory use that pointer style.

If wanna more, refer the third reference.

3.2 GNT object Diagram.

About just only **gameplay phase**, except the others phase.

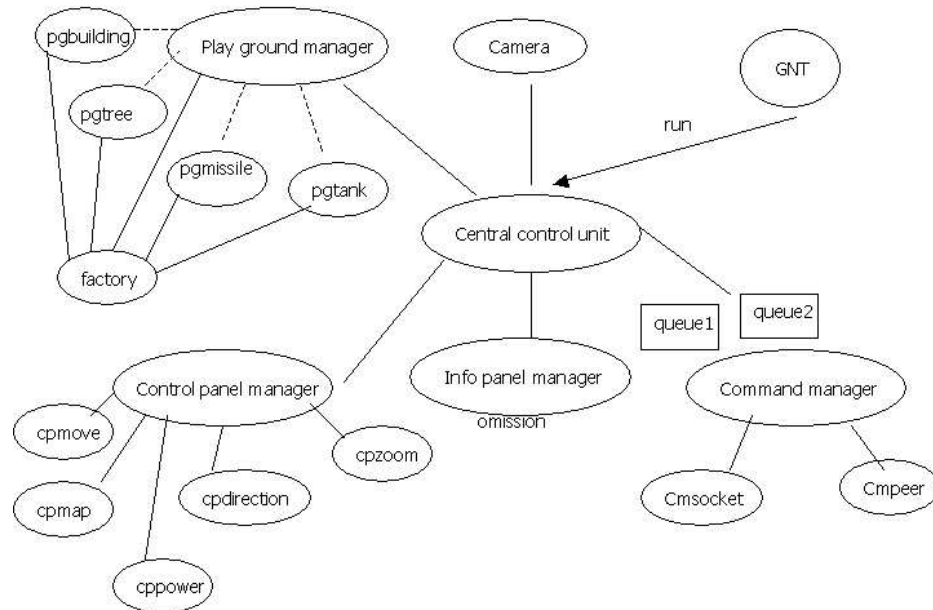


Figure 6: simple object diagram when gameplay phase.

Figure was drawn for easy understanding about GNT. So, Upper figure present important objects except trial objects. There is a little modification for simple.

Detail explanation is under.

3.3 Simple GNT flow.

It is easy to understand, if refer upper figure. Next is importance flow of **Give n Take**

1. **GNT** object initialize all of **Direct related parts**.
2. game play phase started by the **GNT** object call run(in gameloop) of the **centralcontrolunit** object.

3. **central control manager** get commands from **command manager** which is run as another thread.
4. **central control manager** send the commands to **control panel manager** , **info panel manager** and **playground manager** - *ProcessCommand()* routine.
5. **central control manager** call the *Framemove()* routine.
6. **central control manager** call the *Render()* routine.

Additionally, mode of camera setted when *ReceiveCommand()* is called at **central control manager**. The **queue1**, **queue2** is need for command storage and sync. If you change attributes of tank or missile, look at the file in Config directory.

4 Endnotes.

4.1 Author's Note.

I had no idea about D3D and game programming. So, i had to study from base of the game programming by myself. There is no book to teach kindly and easily. The best teacher was the D3D tutorail. Game programming was so difficult in what i had studied before. In result, I think GNT is not enough to release to common people or commercial use. lol

All of this contents is from febace's thought. So, it may be not truth. I do worry about that. :(

4.2 References

- Microsoft DirectX 9.0 tutorial.
- <http://www.andypike.com/>
- http://www.gamasutra.com/features/20000228/ulrich_01.htm