THE DOCUMENT OF METAL GEAR SOLID 2

"PROGRAM" SECTION TRANSCRIPT

Contents

System Structure

The MGS2 system is structured as modules which are divided up as follows:

USER	Programs of the player, enemy, soldiers, doors and bottles, effects.
GAME	A group of functions that can be conveniently used for MGS2 and the program managing the game progress with system program combinations.
SYSTEM	• Script language interpreter library • File/streaming system library • Collision/zone management library • Motion processing library • Miscellaneous multipurpose calculation libraries • Memory management, execution, unit control libraries
KERNEL	• Sound processing library • Thread management wrapper library • CD/DVD control library
SCE LIBRARY	

Each USER program is written as a completed part (Actor). Their activation and messages to them are controlled mainly by our original script language called the GCL. For job efficiency, tasks are divided among the programmers creating each Actor and the Script Unit creating the games by placing each actor.

One characteristic of this system is that when debugging, it constantly measures how long it takes for each Actor's processing. Knowing how long each processing takes allows us to locate which program is "heavy".

This is written in C language. Inline assembler is used in some areas. The source file is managed by CVS, and the compiling is done on Linux. There are 2600 source files (excluding self-generating files), totalling more than 1.15 million lines.

Memory Map

PlayStation 2 has a memory space of 32MB in its Emotion Engine (EE). Of this, each game can use 31MB of memory. MGS2 has allocated them as follows:

4M: Program domain
7M: Permanent data
15M: Work area (including model data, etc. loaded in each stage)
4M: DMA data work for image-drawing
512K: Streaming work
512K: Miscellaneous data

The program is divided up in the permanent portion that is used throughout the game and the non-permanent portion replaced in each stage. By loading only the program used in that stage only, the memory is cut down.

Permanent data includes data of things used in multiple stages. These include player and weapon data. The work area manages the memory internally, while assigned data of each stage and work domains of each program.

The IOP is in charge of sound and CD/DVD access. It functions as a black box that returns data in response to commands from the EE.

Script Language

The GCL (Game Command Language) is the script language designed for the development of MGS. It is a very versatile language. The contents of the script can be divided up into those for “program startup” and “event setting”.

In MGS, programs other than the system are designed as “parts” that can be started up by the Script Unit when putting together a stage. For example, the “player”, “enemy soldier”, “surveillance camera” and “door” are all “parts” that can be assigned positions and parameters freely when starting up the stage. Individual effects are also treated as parts whose parameters can be adjusted freely.

“Event setting” pertains to orders causing events to happen when the player or enemy steps into or out of a predetermined area. For example, “the door will open when the player steps in front of the door” is such an order. In other words, a message is sent so that the “parts” programs that have already been started up perform their functions. These messages are predetermined.

Here is an actual example. The portion beginning with “chara” is what calls for a “parts” program. Such portions are given specific names (“DoorA1” in this case). The parameters assigned are the name of the door model, position, and rotation angle. All of this makes a door show up in a certain position in the game.

```
Chara DOOR door1 ¥
      -kms std_door ¥
      -position 9325,0,-5050
      -rotate 0,2048,0

trap DoorA1_Area SNAKE ¥
      -mask ENTER ¥
      - exec {
            mesg DOOR DoorA1
open
      }
```

The portion beginning with “trap” is what “sets an event”. When “SNAKE” enters the area designated as “DoorA1_Area” by another tool, the execution block beginning with “exec” sends an “open” order to the “DOOR” program called “DoorA1”.

Other situations involve multiple “condition forks” that change what happens depending on the circumstance. The entire game is pretty much composed of combinations of these parts and event sets. It might be fun to play the game trying to guess what kind of script governs each item position, camera behaviour, and individual event you experience.

Collision System

In MGS2, there are collision data in addition to model data. Collision data are classified as “walls” that are always vertical and as “floors” that include everything else. In addition, each is broken down to smaller blocks that allow the slimming down of the subject to be searched in one time.

Each wall and floor is assigned detailed qualities, such as whether or not the enemy can see through it, whether or not bullet marks remain, and sounds of knocks and footsteps.

Unlike MGS1, MGS2 allows weapon attacks in first person view. This necessitated the addition of polygon collision data of the enemy and player. These are searched only when a bullet is fired.

Collision data include many other things such as the designation of areas in which events occur, angle data of the corner view camera, enemy patrol routes, and enemy thought zone data. The actual creation task involves a tool, developed by the Program Unit, that allows GUI setting. Mainly, the script unit uses this tool.

The behaviour is set with the GCL script, based on the data that have been set. In the stage construction of MGS2, the Script Unit first creates with this tool a temporary model with walls and floors only. After it is checked that everything seems to work from a gameplay standpoint, the designers were asked to create actual models based on the data.

The wall information in the radar on the upper right is drawn with the use of this collision information. If you take a close look, you might notice minor differences from the actual background models.

Zone System and Enemy Soldier AI

When an enemy on the upper floor spots the player on the lower floor, the enemy takes the nearest stairs to the lower level, just like someone who knows the area well. This action that sounds quite simple actually involves a lot of thinking. Calculating the shortest distance between two given spots requires a lot of time if done the normal way. In MGS, we resolve this problem with something called the “zone system”.

With the “zone system”, you fill up an irregular shape floor (Diagram A) with many rectangular zones. Lines along which adjacent zones touch each other are recognised as data. When it is determined that there are no obstacles between zones, this information that one can move between these zones is established as a flag. Following this procedure, enemy soldiers determine their routes to their destination. Since this method has in advance the data of whether or not one can move among the given zones, calculation time is decreased considerably.

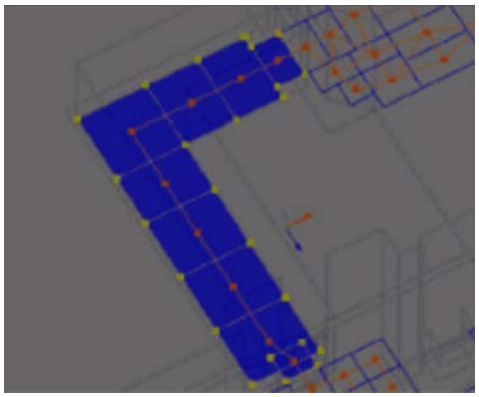

Diagram A

However, there is a problem. When an area is formed of a very complex shape, laying out zones throughout that entire area becomes very difficult. If the enemy steps out of the zones, the enemy cannot determine the route correctly. When this actually happens, the enemy loses track of where he should go. He repeats attempts of changing directions and looks like he is shaking. This phenomenon, often seen during development, was given the name “vibrating soldier”.

In MGS, in addition to the enemy’s normal patrol routes and scouting paths, the player can freely change the position of the enemy by choking and dragging him. This is why a flexible route determination method is required in MGS.

MGS2 Image Drawing System Characteristics

While maintaining the thinking behind the image drawing system of MGS1 on PlayStation, we have designed from scratch a PlayStation 2-specific image drawing system for MGS2.

Model and effect primitives are managed as individual objects in different types depending on what kind they are. Each program only renews the objects display position and vertex information. The image drawing setup is done all together by the system, leading to an overall increase in performance.

The Use of the VRAM and Textures

The PlayStation 2 comes with a 4MB VRAM which we use as follows for MGS2:

Frame Buffer: 2MB (1MB X double buffer) Z Buffer: 1MB Texture Cache: 1MB

This is quite a simple allocation for a PlayStation 2 game. We also use the texture cache for the temporary storage for post effects as well to make efficient use of the limited VRAM.

In MGS2, textures are compiled as a transferrable data unit for each model (background models can reach 1MB, but most models on the average are about 100-500KB in size), only those necessary for the object image-drawing in each image drawing phase are transferred.

Since not all models are drawn on the screen simultaneously, an average of 2 to 3MB of texture transferring takes place per frame in the game. However, the total amount of textures read onto the memory for each stage can go up to 10MB.

Image Drawing Sequence

MGS2 realizes its visuals by preparing the following object types and using them appropriately depending on what it wants to express and how. (names are those used internally)

- **Normal Model:** This refers to the standard MGS2 model. We sometimes refer to it as the single-weight model in contrast with the multi-weight model. Background models with prelighting belong to this type.
  Examples: Background model, etc.
- **Multi-Texture Model:** This refers to a model to which we can apply 3 textures per polygon. The most common combination in MGS2 is the base map, gloss map, and specular environmental map. This is also called the single-weight multi-texture.
  Examples: Normal player model, enemy soldier model, etc.
- **Multi-Weight Multi-Texture Model:** Multi-weight is a function in which we give one vertex the weight of multiple joints. With a normal model in MGS2, 1 polygon is made to work with up to 2 matrices. With a multi-weight model, the model is expanded so that 1 vertex is applied up to 4 matrices and 1 polygon works with up to 8 matrices for assigning weights.
  Examples: Models with facial skeletons for demos, etc.
- **Shadow and Spotlight Casting Processing:** In MGS2, we realize shadows bending along walls with shadow mapping. We create a shadow texture from the light source management object specific to this purpose. This texture is then mapped onto the specific model onto which the shadow is being cast.
- **Shared Model:** A special model drawn in large quantities collectively by setting a certain number of matrices for a model comprised of few polygons – such as a bullet shell. The level of expression is decreased when compared to a normal model. The advantage is that it uses very little memory and the image drawing speed is drastically improved.
  Examples: Bullet shell models, sea lice models, etc.
- **Optical Camouflage Model:** A model with optical camouflage process as seen in the opening of the Tanker Chapter. The screen is once pulled back to the reverse-buffer, and then the buffer is distorted and applied.
- **Patch Curved Surface:** A function where curved surfaces are drawn by designating 4 coordinates on quadrilateral surfaces. In MGS2, curved surfaces are drawn with the appropriate number of polygons by varying how small the polygons are broken down to,

depending on the distance from the camera, etc.
Examples: Ocean surface, water surface in buildings etc.

- **Primitive Object:** Many effects in MGS2 are created with this function. Lines, polygons and 3D sprites can be controlled at the vertex level. By performing a sort in packets of ten vertices when drawing an image, translucent expression with very little collapsing is made possible.

In the MGS2 system, these objects are registered to the DMA buffer collectively. All the image-drawing is done in the background in line with the CPU. Each object is processed in different phases as seen below, and images are drawn in the following order as well.

1. Multi-texture model 1
2. Multi-weight multi-texture model 1
3. Normal model
4. Multi-texture model 2
5. Multi-weight multi-texture model 2
6. Shadow and spotlight casting processing
7. Shared model
8. Optical camouflage model
9. Patch curved surface
10. Translucent model and effect primitive
11. Miscellaneous post 2D effect
12. Miscellaneous 2D image drawing

Image Drawing Processing and Effects

With the advancement from PlayStation to PlayStation 2, the number of polygons and amount of texture that can be shown have increased drastically, leading to increased expression capabilities. What is highly impressive is the level of freedom of the vector unit (VU0, VU1) of the Emotion Engine (EE) along with the incredible fill rate of the graphic synthesizer (GS). The fill rate is actually very much higher than the other consoles of the same generation that were released after PlayStation 2, making itself a notable strength of this console.

With MGS2, we use many translucent effects utilizing the fill rate of the GS. This, along with the vertex processing capability of the PlayStation 2, allows expressions such as the storm that could not be seen in games before.

In MGS2, we made aggressive use of the VU1, making challenges with special image-drawing techniques such as shared models and patch curved surfaces in addition to processes such as 3D calculations and clipping.

Lighting

As for lighting, we are using a system equivalent to that for MGS1. Background models use prelighting, creating in the PlayStation 2 environment each vertex colour from the light source data only once. Because of this, when the light is turned off, recalculation is done to make dynamic changes. As for characters, we use a simple style of 3 parallel light sources and 1 environmental light.

Since light source information is calculated based on the point light source data used for the background prelighting, we have realized lighting that does not differ much from that of the

background. Light source data are set at the point light source. In some places, more than 500 light sources are set in 1 stage.

Streaming System

Streaming is a technology that allows the playing of large and long data by reading and playing data from the CD/DVD in small amounts repeatedly. It is commonly used for movies and voices. As for the MGS series, we have been using a unified format of streaming since MGS1 for polygon demos, voice data and movies.

In MGS2, the following kinds of data are dealt with as streaming data along with time information:

- Sound data
- MPEG2 movie data
- Compressed texture data
- Polygon demo data
- Text display data (font data)
- Codec mode facial motion data

In addition, 2 sets of streaming data can be played simultaneously and independent from each other. This is used for enemy soldier radio communication and the YES/NO answers you can give during the Codec mode.

Sound data, demo data, movie data, although internally separated in files by how they are used, share the same format. They are classified mainly by the data type combinations inside them. Since the amount of voices in MGS2 is enormous, the following data use up the DVD:

Sound data: approx. 1.2Gbytes (580 minutes), demo data: approx.. 1.7GBytes (170 minutes), movie data: 950M (60 minutes).

This technology was developed originally for Policenauts. This allowed the playing of the PCM-recorded BGM and dialogue simultaneously and independently. This technology was ported to other consoles with the conversion of Policenauts. As for Konami JPN titles, this has been used in the Tokimeki Memorial Drama Series. In MGS, this technology has been tuned and made lighter, modified for handling multi-purpose data.

Tools

Various tools were created internally for the MGS2 project.

- Collision data creation/editing tool
- Demo data creation tool
- GCL script compiler
- Model conversion tool
- Facial motion setting tool
- Motion sound effect setting tool
- Motion compression setting tool
- Automatic voice analysis/lip sync creation tool
- Progress management TODO bulletin board
- Data management server system linked to the bulletin board
- Data construction script
- Streaming creation tool

Etc. etc.

Since MGS2 was a very large project, the ease of use of these tools had a direct impact on the efficiency of tasks. These tools were then modified repeatedly while being used to meet the need of those who used them.

Not all tools ended up meeting everyone's needs. However, we will most definitely benefit from what we learned through their development and use.

Data Management

One of our goals with MGS2 was efficient data management. This was because we faced many problems when dealing with large data volumes for MGS1.

The MGS2 data consists of approximately 3,300 models, 53,000 motion patterns including those created automatically, and about 10,000 voice patterns. A total of 70 GBytes of data ended up being registered on the server. In order to manage all of this efficiently, we constructed a CGI management system with a data-upload-type bulletin board basis.

Each creator registers his/her data through a browser-style interface, and the server handles the conversion to a format that can be used in the PlayStation environment. Those who wish to use the data will create a downloadable list script for each stage and then use a tool that automatically constructs the data according to that script.

Thanks to this, the server handled data formatting fixes efficiently, preventing most cases of accidental rewriting of data. The management of source codes and GCL script is done quite well with use of the CVS.

However, due to the overall increase of data, the overall time for all construction has increased greatly. Currently, it takes more than 2 hours to renew and update all MGS2 data and programs. This is a pain in the neck especially towards the end of the development cycle, when modification and confirmation becomes frequent.