



ANOTHER WORLD

INTRODUZIONE

- ▶ Scritto da Eric Chahi per Delphine Software nel 1991
- ▶ Pubblicato originariamente per Amiga e Atari ST, convertito per svariate piattaforme (MS-DOS PC, Apple II GS, Nintendo SNES, Sega MegaDrive, Sega MegaCD, Panasonic 3DO, Nintendo Gameboy Advance, ecc...)
- ▶ Platform game, innovativo per aspetti grafici e di cinematica (sia real-time che in cutscene)



INTRODUZIONE

- ▶ Versione PC MS-DOS distribuita su un singolo floppy da 1,44MB e richiede poco meno di 600KB di memoria RAM durante l'esecuzione
- ▶ Aspetti importanti di questo gioco sono nel engine e nella grafica
- ▶ Cuore dell'esecuzione è una virtual machine che interpreta del bytecode e genera sequenze cinematografiche vettoriali in tempo reale
- ▶ Source code originale non è mai stato rilasciato, ma è disponibile un reverse engineering della versione DOS. Inoltre è stato creato un moderno bytecode interpreter in C++.
- ▶ .exe di DOS è molto piccolo perché comprende solo VM. Porting su piattaforme diverse consiste in riscrittura della VM, mentre bytecode rimane invariato.
- ▶ Compiti della VM:
 - host per bytecode
 - system call (disegno parti grafiche, riproduzione suoni, ecc...)

GRAFICA

- ▶ Tutto formato da poligoni (niente elementi bitmap!)
- ▶ Utilizzo di tre diversi framebuffer, di cui uno per lo sfondo: esso è disegnato una sola volta per ogni scena e riutilizzato (memcpy) negli altri due framebuffer



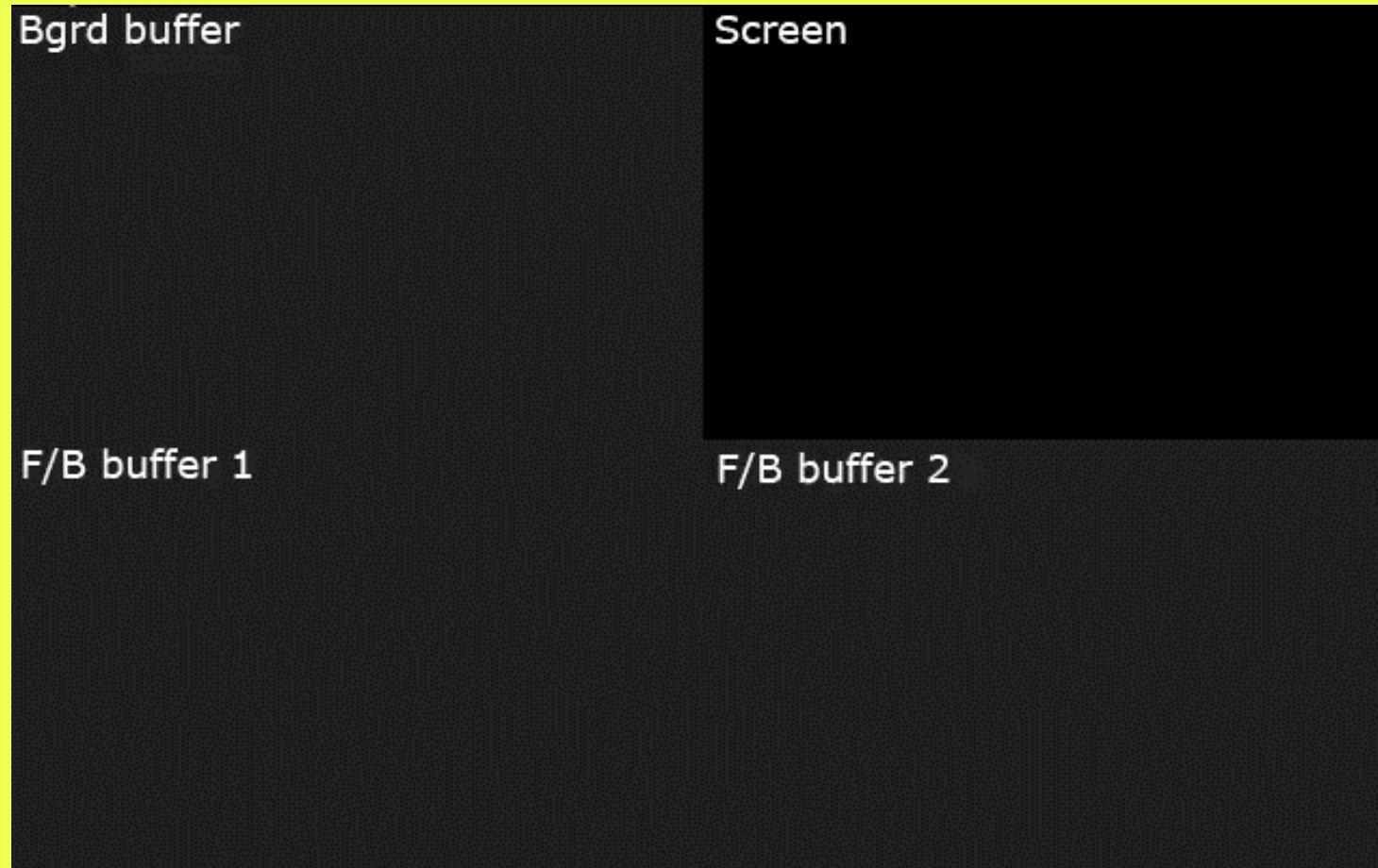
GRAFICA

- ▶ Tutto formato da poligoni (niente elementi bitmap!)



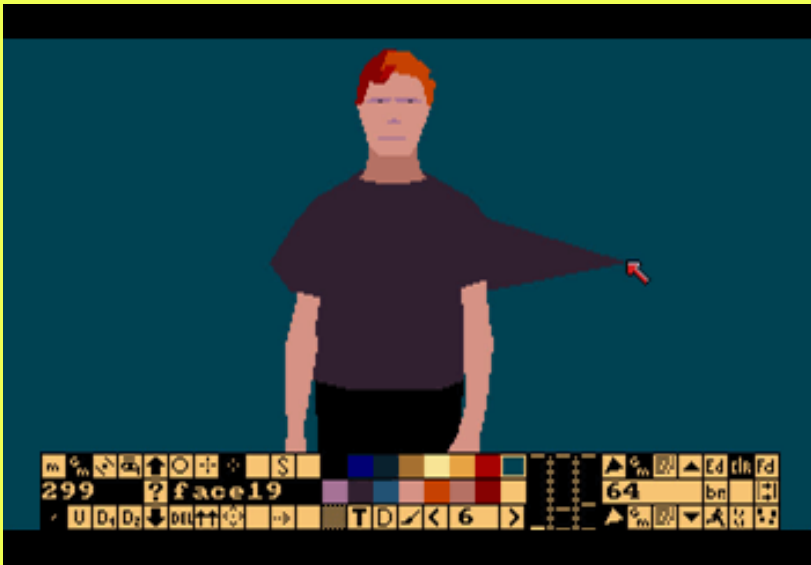
GRAFICA

- ▶ Triplo framebuffer



GRAFICA

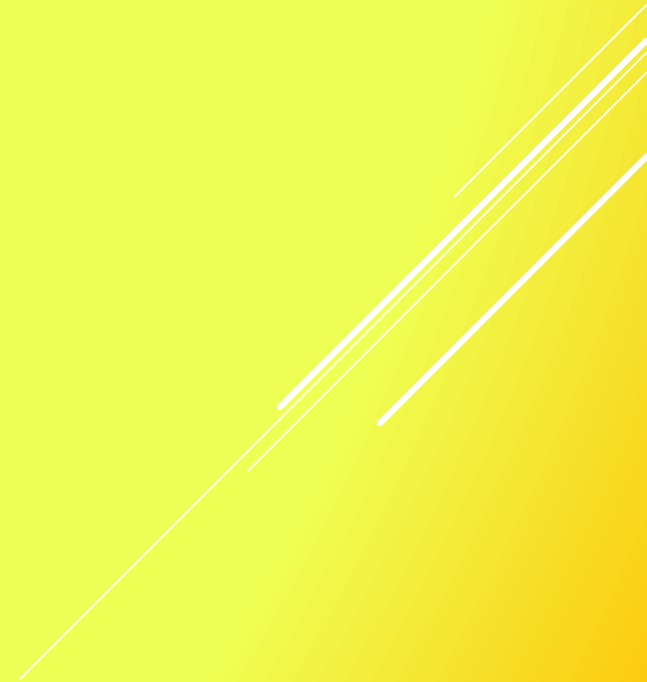
- ▶ Polygon assembling: *“Each visual display unit of the game was composed by many polygons, so it was essential to gather many polygons in one item, to facilitate their manipulation. For example, in a character:”*



- ▶ Hierarchical structure of display: *“The polygon groups could be put together in turn to form another item. This hierarchical structure avoided redundancy through the creation of a priority system. For example, making a specific group for the head of the hero could be re-used in all phases of animation.”*

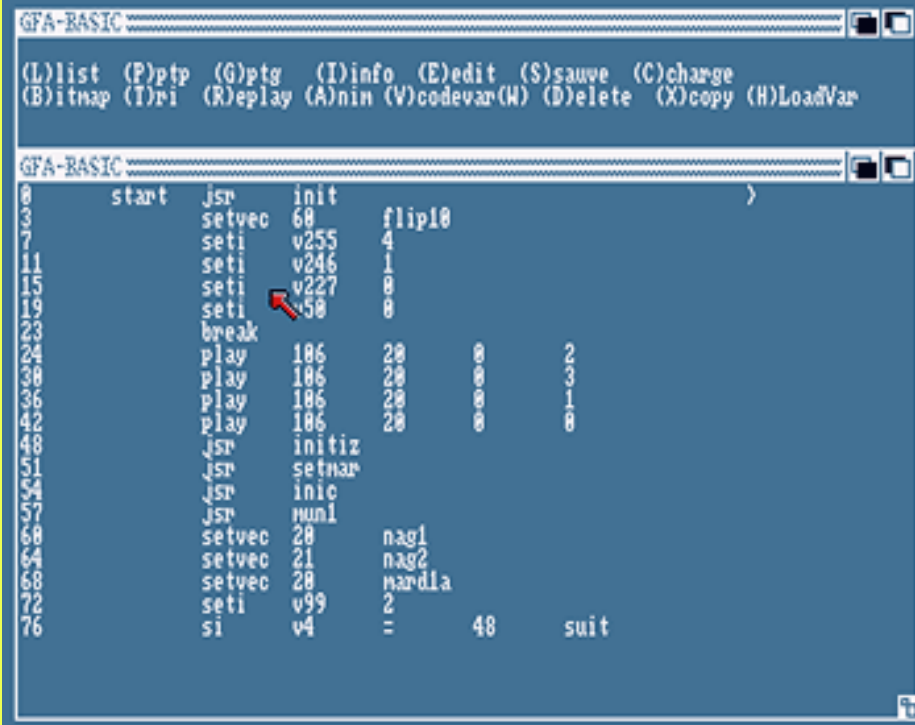
GRAFICA

- ▶ Animazioni create con tecniche di rotoscoping: ripresa di veri attori e copia di ogni fotogramma su celluloide
- ▶ Nell pratica: sequenze riprese con camcorder, riprodotte con un VCR e mandate in input ad un Amiga tramite Genlock per poi essere disegnate manualmente nel polygon editor



ARCHITETTURA

- ▶ “The game logic should be coded in a language independent from all platforms, without needing compiling or data conversion. So I naturally orientated myself towards the creation of a script interpreter. I developed a mini-language structured in 64 independent execution channels and 256 variables. The instruction set of the language is very reduced, at around only thirty "words". And within this the game logic has been programmed, including the joystick instructions.”

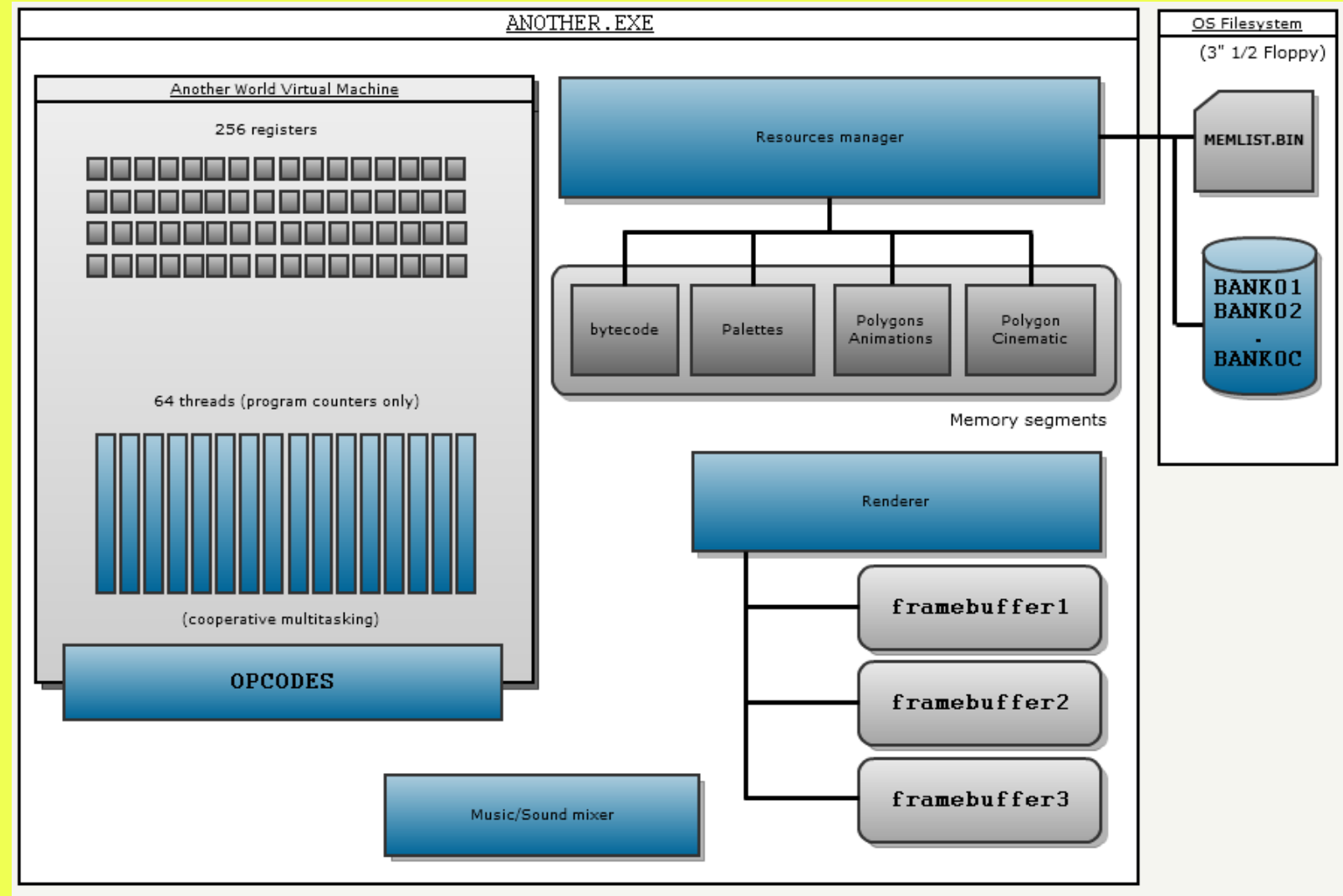


```
GFA-BASIC
(L)list (P)ptp (G)ptg (I)info (E)edit (S)sauve (C)charge
(B)itnap (T)ri (R)eplay (A)nin (V)codevar(W) (D)elete (X)copy (H)LoadVar

GFA-BASIC
0      start  jsr  init
3      setvec 60  flip10
7      seti  v255 4
11     seti  v246 1
15     seti  v227 0
19     seti  v50  0
23     break
24     play  106  20  0  2
30     play  106  20  0  3
36     play  106  20  0  1
42     play  106  20  0  0
48     jsr  initiz
51     jsr  setnar
54     jsr  inic
57     jsr  hum1
60     setvec 20  nag1
64     setvec 21  nag2
68     setvec 20  wardla
72     seti  v99  2
76     si  v4  =  48  suit
```

ARCHITETTURA

- ▶ 4 moduli:
 - Virtual Machine
 - Resource Manager
 - Sound/Music Mixer
 - Renderer



ARCHITETTURA

- ▶ L'esecuzione del gioco è strutturata in 64 «channel» indipendenti e 256 variabili comuni (esempio più vicino è il multi-thread cooperativo)
- ▶ Ogni channel esegue una parte specifica di codice, (x es. un channel si occupa del movimento e della logica di un personaggio, un altro channel del suo rendering, un altro disegna degli uccelli sullo sfondo, un altro ancora trigger un'animazione conseguente un particolare evento).
- ▶ I channel sono eseguiti in ordine e l'istruzione `break` indica di andare al canale successivo
- ▶ Quando si raggiunge il canale 64, il frame corrente viene mostrato e il ciclo riparte
- ▶ Ogni channel può settarne un altro con l'istruzione `setvec`

ARCHITETTURA

Instructions related to media, graphics, sound, palette:

Load "file number"

Loads a file in memory, such as sound, level or image.

Play "file number" note, volume, channel

Plays the sound file on one of the four game audio channels with specific height and volume.

Song "file number" Tempo, Position

Initialises a song.

Fade "palette number"

Changes of colour palette.

Clr "Screen number", Color

Deletes a screen with one colour.
Ingame, there are 4 screen buffers.

Copy "Screen number A", "Screen number B"

Copies screen buffer A to screen buffer B.

Show "Screen number" :

Displays the screen buffer specified in the next video frame.

SetWS "Screen number" :

Sets the work screen, which is where the polygons will be drawn by default.

Spr "object name" , x, y, z

In the work screen, draws the graphics tool at the coordinates x,y and the zoom factor z. A polygon, a group of polygons...

Text "text number", x, y, color

Displays in the work screen the specified text for the coordinates x,y.

Variables and their manipulation :

Set.i variable, value

Initialises the variable with an integer value from -32768 to 32767.

Set variable1,variable2

Initialises variable 1 with variable 2.
 $\text{Variable1} = \text{Variable2}$

Addi Variable, Value

$\text{Variable} = \text{Variable} + \text{Integer value}$

Add Variable1, Variable2

$\text{Variable1} = \text{Variable 1} + \text{Variable2}$

Sub Variable1, Variable2

$\text{Variable1} = \text{Variable1} - \text{Variable2}$

Andi Variable, Value

$\text{Variable} = \text{Variable AND valeur}$

Ori Variable, Value

$\text{Variable} = \text{Variable OR valeur}$

Lsl Variable, N

Makes a N bit rotation to the left on the variable. Zeros on the right.

Lsr Variable, N

Makes a N bit rotation to the right on the variable.

Instruction branch :

Jmp Adresse

Continues the code execution at the indicated address.

Jsr Address

Executes the subroutine located at the indicated address.

Return

End of a subroutine.

Conditional instructions :

Si (Variable) Condition (Variable ou value) jmp adresse

Conditional branch,

If (=Si) the comparison of the variables is right, the execution continues at the indicated address.

Dbra Variable, Address

Decrements the variable, if the result is different from zero the execution continues at the indicated address.

At last but not least, structural instructions :

Setvec "numéro de canal", adresse

Initialises a channel with a code address to execute

Vec début, fin, type

Deletes, freezes or unfreezes a series of channels.

Break

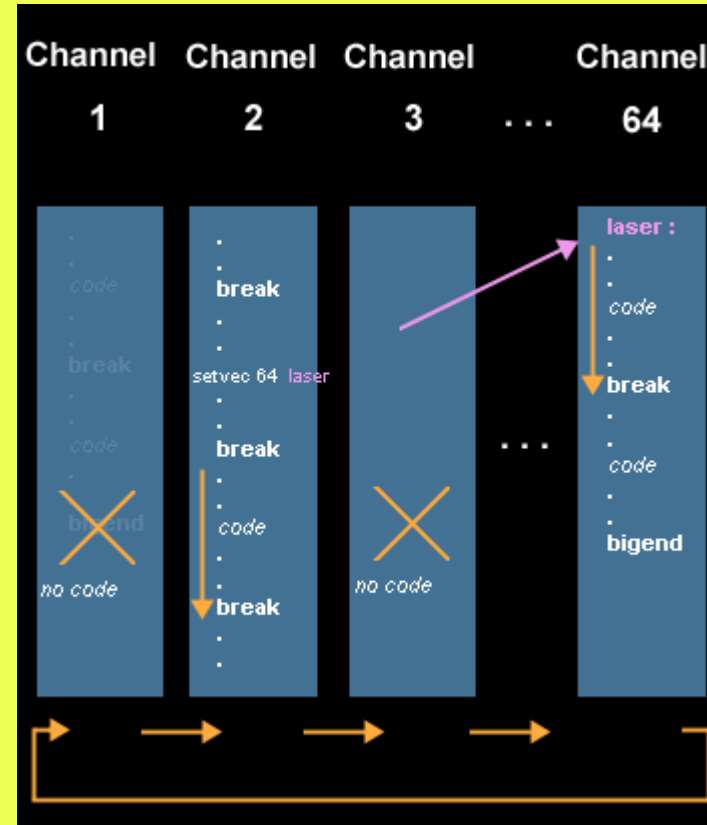
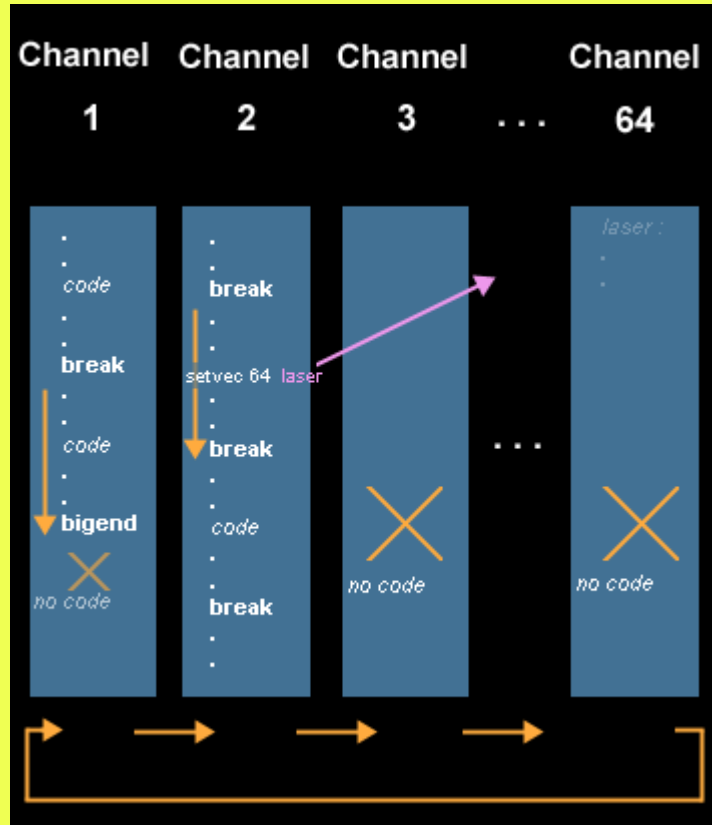
Temporarily stops the executing channel and goes to the next.

Bigend

Permanently stops the executing channel and goes to the next.

ARCHITETTURA

- Esempio: channel 2 gestisce logica personaggio, che decide di sparare un laser e inizializza canale 64 per disegnare e gestire sparo:



ENGINE IN C++

- ▶ Implementazione completa del engine in C++ (originale era in assembly):
<https://github.com/fabiensanglard/Another-World-Bytecode-Interpreter>
- ▶ Simile ad engine originale per gestione risorse e memoria (niente malloc)
- ▶ Renderer, audio, controlli tramite SDL

ENGINE IN C++

▶ main.cpp	
bank.cpp	lettura/unpack dei file bank*
engine.cpp	classe principale
file.cpp	
intern.h	
mixer.cpp	sw mixer, play PCM
parts.cpp	
resource.cpp	lettura da file bank*
serializer.cpp	
sfxplayer.cpp	
staticres.cpp	opcode table, font, stringhe di testo
sysImplementation.cpp	link con SDL
util.cpp	
video.cpp	sw renderer
vm.cpp	classe della vm

ENGINE IN C++

► Partiamo dai dati... Resource.cpp

```
► struct Resource {  
    enum {  
        MEM_BLOCK_SIZE = 600 * 1024    //600kb total memory consumed  
                                           //(not taking into account stack and static heap)  
    };  
    MemEntry _memList[150];  
    uint8_t *_vidBakPtr, *_vidCurPtr;  
    uint8_t *segBytecode;  
    uint8_t *segCinematic;  
}
```

MemEntry è array di resource; ogni resource identificata da un id univoco. A startup, viene aperto file MEMLIST.BIN e vengono letti questi record:

```
typedef struct memEntry_s {  
    int bankId;  
    int offset;  
    int size;  
    int unpackedSize;  
} memEntry_t;
```

ENGINE IN C++

- ▶ Quando la vm richiede una risorsa, il resource manager:
 - apre il file BANK dove è memorizzata
 - salta a offset e ne legge size
 - se `size != unpackedSize`, la risorsa deve essere scompattata

Tipi di risorsa:

- ▶

```
enum ResType {  
    RT_SOUND           = 0,  
    RT_MUSIC           = 1,  
    RT_POLY_ANIM       = 2, // full screen video buffer, size=0x7D00  
    RT_PALETTE         = 3, // palette (1024=vga + 1024=ega), size=2048  
    RT_BYTECODE        = 4,  
    RT_POLY_CINEMATIC  = 5  
};
```

ENGINE IN C++

- ▶ main.cpp [semplificato]

- ▶

```
//extern System *System_SDL_create();  
extern System *stub ;//= System_SDL_create();  
  
int main(int argc, char *argv[]) {  
    Engine* e = new Engine(stub, dataPath, savePath);  
    e->init();  
    e->run();  
  
    delete e;  
  
    return 0;  
}
```

- ▶ Semplice, no?

ENGINE IN C++

- ▶ Classe Engine

- ▶ Contiene una istanza di:

```
System          *sys;  
VirtualMachine  vm;  
Mixer           mixer;  
Resource        res;  
SfxPlayer       player;  
Video           video;
```

- ▶ Engine::init() chiama i rispettivi membri funzione ::init() di ogni oggetto

```
▶ Engine::run() {  
    while (!sys->input.quit) {  
        vm.checkThreadRequests();  
        vm.inp_updatePlayer();  
        processInput();  
        vm.hostFrame();  
    }  
}
```

ENGINE IN C++

- ▶ Classe Video
- ▶ Contiene i tre framebuffer: sono array lineary di uint8_t e tutte le funzioni per disegnare poligoni in essi
- ▶

```
struct Video {  
    enum {  
        VID_PAGE_SIZE  = 320 * 200 / 2    // 16 colori => 4 bit per pixel!!!  
    };  
    // _curPagePtr1 is the backbuffer  
    // _curPagePtr2 is the frontbuffer  
    // _curPagePtr3 is the background builder.  
    uint8_t *_curPagePtr1, *_curPagePtr2, *_curPagePtr3;  
}
```
- ▶ Classi Sfxplayer e Mixer trattano audio – non le vedremo ☹

ENGINE IN C++

- ▶ Infine, classe Vm [quella che tutti aspettavamo!]

- ▶ Vm.h

- ▶

```
#define VM_NUM_THREADS 64
#define VM_NUM_VARIABLES 256
```
- ▶

```
//For threadsData navigation
#define PC_OFFSET 0
#define REQUESTED_PC_OFFSET 1
```
- ▶

```
//For vmIsChannelActive navigation
#define CURR_STATE 0
#define REQUESTED_STATE 1
```
- ▶

```
enum ScriptVars {
    VM_VARIABLE_RANDOM_SEED          = 0x3C,
    VM_VARIABLE_LAST_KEYCHAR          = 0xDA,
    VM_VARIABLE_HERO_POS_UP_DOWN     = 0xE5,
    VM_VARIABLE_MUS_MARK              = 0xF4,
    VM_VARIABLE_SCROLL_Y              = 0xF9, // = 239
    VM_VARIABLE_HERO_ACTION           = 0xFA,
    VM_VARIABLE_HERO_POS_JUMP_DOWN    = 0xFB,
    VM_VARIABLE_HERO_POS_LEFT_RIGHT   = 0xFC,
    VM_VARIABLE_HERO_POS_MASK         = 0xFD,
    VM_VARIABLE_HERO_ACTION_POS_MASK  = 0xFE,
    VM_VARIABLE_PAUSE_SLICES          = 0xFF
};
```

ENGINE IN C++

```
► struct VirtualMachine {

    Mixer *mixer;
    Resource *res;
    SfxPlayer *player;
    Video *video;
    System *sys;

    int16_t vmVariables[VM_NUM_VARIABLES];
    uint16_t threadsData[NUM_DATA_FIELDS][VM_NUM_THREADS];
    // This array is used:
    //      0 to save the channel's instruction pointer
    //      when the channel release control (this happens on a break).
    //      1 When a setVec is requested for the next vm frame.
    uint8_t vmIsChannelActive[NUM_THREAD_FIELDS][VM_NUM_THREADS];

    Ptr _scriptPtr;           // Puntatore al bytecode
    uint8_t _stackPtr;        // stack pointer
```

ENGINE IN C++

► struct VirtualMachine [continuazione]

```
// The type of entries in opcodeTable. This allows "fast" branching
typedef void (VirtualMachine::*OpcodeStub)();
static const OpcodeStub opcodeTable[];

void op_movConst();
void op_mov();
[ ... ]
void op_updateMemList();
void op_playMusic();

void initForPart(uint16_t partId);
void setupPart(uint16_t partId);
void checkThreadRequests();
void hostFrame();
void executeThread();

void inp_updatePlayer();
}
```

ENGINE IN C++

- ▶ Ma come sono fatti questi opcode? Vediamone uno facile...

```
▶ void VirtualMachine::op_add() {  
    uint8_t dstVariableId = _scriptPtr.fetchByte();  
    uint8_t srcVariableId = _scriptPtr.fetchByte();  
    debug(DBG_VM, "VirtualMachine::op_add(0x%02X, 0x%02X)", dstVariableId, srcVariableId);  
    vmVariables[dstVariableId] += vmVariables[srcVariableId];  
}
```

- ▶ Mi sembra il caso di vedere come avvenga l'esecuzione... Prendiamo in mano ancora il Engine::run() :

```
▶ Engine::run() {  
    while (!sys->input.quit) {  
        vm.checkThreadRequests();  
        vm.inp_updatePlayer();  
        processInput();  
        vm.hostFrame();  
    }  
}
```

ENGINE IN C++

- ▶ Engine::run()

- ▶

```
Engine::run() {  
    while (!sys->input.quit) {  
        vm.checkThreadRequests();  
        vm.inp_updatePlayer();  
        processInput();  
        vm.hostFrame();  
    }  
}
```

- ▶ `vm.checkThreadRequests();`

Controlla se è stato chiesto un cambio di scena; controlla quali thread sono attivi nel frame corrente e modifica `threadsData[][]` in accordo

- ▶ `vm.inp_updatePlayer();`

Legge tastiera e modifica le variabili di stato del giocatore in `vmVariables[]`

- ▶ `vm.hostFrame();`

Esegue la vm per ogni thread attivo nel frame corrente: ciclo for 0..63. Thread inattivi vengono saltati; thread attivi vengono eseguiti dalla funzione `VirtualMachine::executeThread()`

ENGINE IN C++

- ▶ VirtualMachine::executeThread() : siamo quasi alla fine!

```
▶ void VirtualMachine::executeThread() {  
    while (!gotoNextThread) {  
        uint8_t opcode = _scriptPtr.fetchByte();  
        if (opcode & 0x80) {  
            [ casi draw trattati a parte ]  
            continue;  
        }  
        if (opcode & 0x40) {  
            [ casi draw trattati a parte ]  
            continue;  
        }  
        if (opcode > 0x1A) {  
            [ errore! ]  
        }  
        else {  
            (this->*opcodeTable[opcode]) ();  
        }  
    }  
}
```

- ▶ Invece di array di puntatori a funzione si poteva implementare switch..case

ENGINE IN C++

► Manca qualcosa...

```
► Engine::run() {  
    while (!sys->input.quit) {  
        vm.checkThreadRequests();  
        vm.inp_updatePlayer();  
        processInput();  
        vm.hostFrame();  
    }  
}
```

ENGINE IN C++

► Manca qualcosa...

```
► Engine::run() {  
    while (!sys->input.quit) {  
        vm.checkThreadRequests();  
        vm.inp_updatePlayer();  
        processInput();  
        vm.hostFrame();  
        video.showFrame();    // O qualcosa di simile...  
    }  
}
```



ANOTHER WORLD

- ▶ Bibliografia:
- ▶ <https://github.com/fabiensanglard/Another-World-Bytecode-Interpreter>
- ▶ http://fabiensanglard.net/anotherWorld_code_review/index.php
- ▶ http://www.anotherworld.fr/anotherworld_uk/another_world.htm