CHAPTER 7

CryVE:

Modding the CryEngine2

to create a

CAVE System

Marija Nakevska
Alex Juarez
Jun Hu

CryVE: Modding the CryEngine2 to create a CAVE system

Marija Nakevska, Alex Juarez, Jun Hu,
Dept. of Industrial Design, Eindhoven University of Technology, Eindhoven, The Netherlands
m.nakevska@tue.nl, acordova@tue.nl, j.hu@tue.nl

Game engines and game mods

Game engines

A *game engine* is a multiplatform middleware software system that facilitates game development. Game engines include functionality needed to develop game applications by using a flexible and reusable software platform. This helps to reduce costs and complexity in game development. Fundamental components of modern game engine platforms are a rendering engine for 2D or 3D graphics, and a physics engine for accurate physics simulation. Other common features are collision detection, sound, scripting, animation, artificial intelligence, networking, streaming, etc.

An example application of game engines is a *first person shooter (FPS) game. A first person shooter (FPS)* game is a video game in which the players see the world from the eyes of their characters and the game revolves primarily around eliminating or disabling other entities in the game world. A first person shooter game engine is a game engine specialized for simulating 3D environments for use in a first-person shooter video game.

Game engines are often used for other kinds of interactive applications with real-time graphical needs such as marketing demos, architectural visualizations, training simulations, and modeling environments. As game engine technology has matured and become more user-friendly, the application of game engines has also broadened in scope and game engines are used in visualization, training, medical and military simulation applications.

Game mods

Game engines allow designers to create new game behaviors and graphics by plugging into reusable architectures that handle polygon rendering, camera control, lighting and so on. Game engines usually come with scripting

languages that allow users to modify behaviors, create new worlds or modify existing games into completely new ones. Game modifications (generally called mods) are software plugins made by the general public or developer. They can include new items, characters, models, textures, levels, story lines, music and other game mods. Game mods are not standalone software and require the user to have the original game release in order to run.

Commercial game developers have promoted game mods to the community in an "open architecture" approach where partial codes and updates are available to manipulate the game engine. User-created content is usually gathered in knowledge bases for game playing, editing, level (or map) building and distribution. With creating new content and knowledge bases, the community extends and enriches game play experiences. The community also provides free support and tutorials for code alterations, hints for obtaining performance benefits of each other.

Cry Engine

CryEngine is a multiplatform game engine originally developed by Crytek. CryEngine has opened up many possibilities for game designers with its ability to handle extremely large photorealistic interior and vast outdoor spaces while still supporting the scene details (Seeley, 2007).

CryEngine supports a number of features that are useful for creating immersive and realistic games and virtual environments. The necessary development tools are integrated with the engine itself, including *CryEngine Sandbox* world editing system and the Mod SDK that are available as free downloads.

Crysis Mod SDK

Crysis Mod SDK contains tools; assets and game source code to help modders (gamers, hobbyists, and game developers) create their own game mods. The kit includes everything needed to set up a custom modification for *Crysis* and *CryEngine2*, including sample code, tools, mods and assets.

CryEngine Sandbox Editor gives full real-time control to the developers over their multi-platform creations. It introduces a "What You See Is What You Play" (WYSIWYP) system where games can be produced and played immediately. The Sandbox editor is used to create levels for *CryEngine*-based games, and tools are provided within the software to facilitate scripting,

animation and object creation.

The foundation for a virtual world is laid by creating a new *map (level)*. Creation and editing of the level and the terrain is facilitated with tools for creating landscape, surface and textures, to which different effects can be applied.

A *Flow Graph* is a simple visual programming system in *CryEngine* that gives designers an intuitive interface to create and control events, triggers, game logic, effects, and sound design. The *Flow Graph System* facilitates the building of complex levels without the need of writing code.

The *Track View* is an editing tool embedded in the *Sandbox Editor* that allows creating cut scenes for making interactive movie sequences. Creating cinematic cut scenes and scripted events can be done in-game by setting a sequence of objects, animations and sounds in a scene which can then be triggered automatically by character interaction. Sequences created with *Track View* can also be triggered in-game with a specific *Flow Graph* node.

CAVE systems

A CAVE system is an immersive virtual reality interface usually consisting of an enclosed environment that surrounds the user with projected images. CAVE systems appeared at the beginning of 1990's as a visualization tool for virtual reality environments that utilized an array of large projection screens, resembling a room whose walls, ceiling and floor showed images specially designed to provide the experience of "immersion" in the virtual world (Cruz-Neira et al., 1992). Since then, several variations of this setup have been developed, examples including asymmetric screens, portable CAVEs with only two walls (Sauter, 2003), arrangements in "U" shape configurations, and semi-spherical screens.

In general, multi-screen immersive systems typically require one or two video outputs for each screen and simultaneously utilize several interaction devices. The capabilities of the CAVE hardware must be assessed before attempting to develop the simulation, as running multiple projections simultaneously requires a fast network and substantial graphical processing. If the network bandwidth is unacceptably low or if the processor speed is inadequate, a distributed CAVE simulation might not be feasible.

Software for CAVE systems has also evolved from research-oriented, custom-

made applications that modified the image aspect ratio and display quality, into powerful software tools. Existing solutions are capable of performing high-end 3D modeling and rendering, incorporating multiprocessing and acting as "glue" to other virtual reality components. Other devices can also be incorporated to the system, such as head mounted displays, 3D glasses, pressure and temperature sensors, etc. (Cruz-Neira et al., 2002).

Recently the cost and performance of CAVE systems received a boost with the appearance of game-engine-based virtual reality systems. These systems used commercially available software packages - game engines - that provide advanced simulation and graphics. The most widely known example of this kind of system is CAVEUT (Jacobson & Lewis, 2005, Jacobson 2005), a CAVE based on an extension of the Unreal Engine 2.5 (Unreal technology, 2010). CAVEUT public application programming interface (API) enables users with limited programming experience to build a fully functional CAVE, without having to modify the internal workings of the game engine itself.

A similar CAVE system was reportedly built using the Half-Life Engine (Schou et al. 2007), however, the capabilities, implementation requirements, and costs of such a system were not clearly explained, nor was it clear whether it could currently compete with the graphics, physics and 3D model quality of the Unreal Engine 2.5.

In any case, the constant development of games and game engines results in even more accurate and realistic, state of the art simulation and rendering of virtual environments, all done using commodity hardware. At the same time, current game engines offer the possibility to easily modify and extend the content and behavior of characters and virtual environments through "in-game" editors and public APIs.

Low-cost implementation of a CAVE system

In previous publications, we mentioned that there is a gap in qualitative and quantitative terms between commercial and open source solutions. This gap is the difference between systems that provide stunning visuals, state of the art modeling, rendering and visualization, and continuous support at high costs; and systems that provide cost effective solutions, with more limited visuals, development capabilities, and overall user experience (Nakevska et al., 2011; Juarez et al. 2010).

As a rule of thumb, commercial solutions like the i-Space (Barco Solutions, 2010) or Apollo (HoloVis Cave solutions, 2010) CAVE systems offer

accurate and realistic experience customized to the needs of the client, with extensive application and installation support, unfortunately, the cost can be prohibitive. On the other hand, open source, non-commercial solutions like AGAVE (Leigh et al., 2001) and CAVEUT (Jacobson et al., 2005) offer cost-effective setups that are in many cases implemented only at education institutions, using outdated technology, with limited development and scarce support for potential users.

A low-cost implementation is needed to fill in the gap between commercial and open source systems.

The main characteristics of such an implementation would be:

- A low-cost, full-size physical construction. While there are CAVE systems of reduced dimensions at affordable prices, the truly immersive experience comes from a physical setup that can accommodate at least one person standing comfortably, allowing for some space to move and interact with the system.

- Easy to setup, maintain and extend. A system that requires the intervention of experts for even the simplest tasks are unlikely to be more efficient than those where a dedicated enthusiast can get satisfying results. This means that the use of standard and simple components, as well as uncomplicated methods and mechanisms to modify and extend the CAVE are preferred. Furthermore, the existence of an active development community that provides support and continuously improves and extends the software, is also an important factor.

- Realistic immersive experience. A successful CAVE system must provide a believable experience to a spectator, presenting a visually rich virtual environment. The available virtual environment development toolkits provide only a subset of the tools needed to build complete virtual worlds, reusing the computer game technology is alternative for building realistic virtual worlds featuring user-friendly interaction and the simulation of real world.

CryVE CAVE System

CryEngine automatic Virtual Environment (CryVE) is a CAVE system based on the game engine *CryEngine 2. CryEngine 2* is game engine developed by the company Crytek. Crytek is founded in 1999, after releasing numerous demos of the game X-Isle, it evolved to the game *Far Cry* and the *CryEngine* that the game uses. In 2007 the game Crysis with the CryEngine 2 was released. The game *Crysis Warhead* as an expansion of *Crysis* was released in 2008 as a PC-exclusive game. The game engine *CryEngine 3* is released on October 2009, in August 2011 Crytek has released *CryEngine 3 Free SDK* package. *CryEngine 3* is free for educational and non-commercial use. Sample assets are included with the *Free SDK*, but also artists, animators and audio engineers can design and export assets to the *CryEngine.*

The software architecture uses the multiplayer features of a *CryEngine* computer game to build projections for the different sides of the installation. The system architecture is similar to that of CAVEUT: multiplayer instances of a *CryEngine 2* game are started on all computers in the system. Computers are connected to each other through a network hub, with one of them acting as a server (master) while the rest are game clients (slaves). The server can control the in-game action (walking, jumping, shooting, etc.) while the clients provide the extra "cameras" that complete the peripheral view required by the CAVE, aligning and synchronizing themselves to the pose and motion of the master.

Finally each computer renders its piece of the virtual world to the corresponding projector and projection screen (see *Figure 1*). In principle any computer game that is based on the *CryEngine 2* can be used in a CryVE setup. Examples of games using *CryEngine 2 are Crysis, Crysis Warhead, Entropia Universe* and *Blue Mars*.

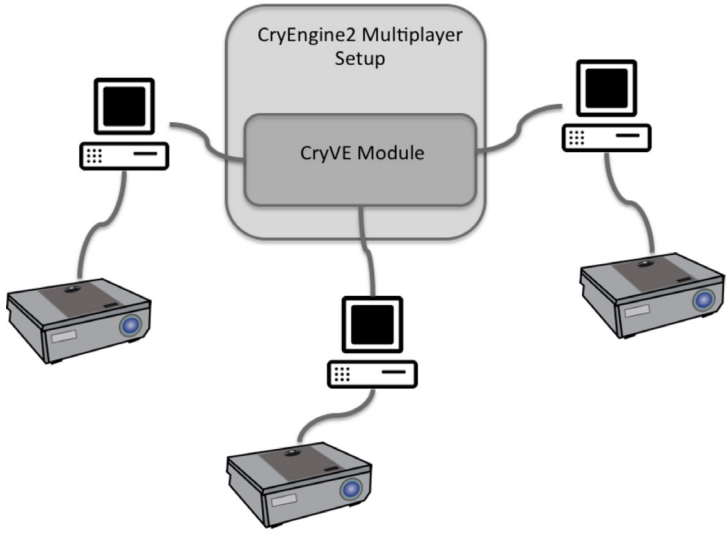

Figure 1; CryVE system architecture.

Physical Setup

The CryVE system physical setup is an arrangement of screens resembling a cubic room, with the projection done from the outside of the room. This allows viewers to move around inside without creating undesired shadows on the projection screens. The installation was built around an aluminum frame that held a plastic white translucent canvas as seen in *Figure 2a*.

Figure 2; Physical installation of the CryVE system.
(Left) Steel frame holding the canvas
(Right) Projectors mounted 4.0 meters above ground

Each side of the room measures 3 X 3 meters and five of the six walls are projecting screens, leaving out only the floor. Each projector is controlled by a single computer, which in turn is connected to a computer network. The first prototype that we created consisted of only three faces of the cube used as projection screens, resulting in a "U" CAVE configuration.

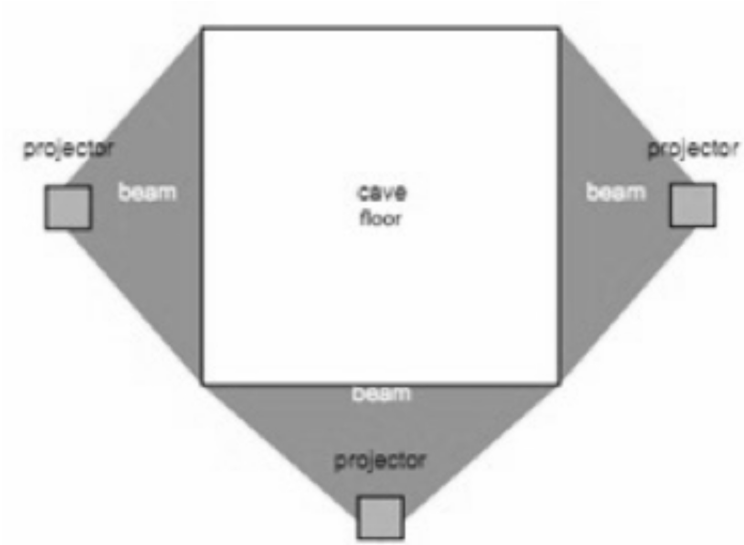

Figure 3; Physical installation of the CryVE system.

The room enclosing the CAVE installation offered a maximum of 2.2 meters of space on each projecting side. Due to this limitation in the physical space available, Hitachi ED-A100 XGA projectors were used. These projectors offer the advantage of a short throwing distance and easy image adjustment to cover the square canvas of the projection screen, albeit at a higher cost than normal projectors. Furthermore, the back projection nature of these devices allowed us to mount them at 4.0 meters above ground outside of the room, enabling free transit around the CAVE without undesired shadows appearing on the canvas (see *Figure 2b*). Each projector was then connected to a computer that controlled one of the projected faces of the room.

Software setup

The CryVE software implementation consists of three components: a game modification (mod), a game Flow Graph and a modified multiplayer map. A *CryEngine 2* game mod is a piece of code (usually written in C++) that can access the low-level data structures and API of the game engine, and extends

its functionality by modifying the behavior and appearance of characters, and even the gameplay itself.

The CryVE mod component is in charge of deciding if a specific instance of a game is dedicated as a server or client in the CryVE setup. If the current instance is a master, the mod sends a signal to potential clients (other slaves) that it is available to connect to. If the instance is a client, the mod obtains the camera gaze of the master and aligns the camera view of the client accordingly.

CryVE mod and programming environment of CryEngine

Level setup

Creating the foundation for a virtual world starts with the creation of a new map (level). The *Sandbox editor* facilitates the process of creation of new maps. Using the option *File>New*, the map is created in new folder which contains all the required files. The most important file is the .cry file, which contains all the major information for the editor. Creating and editing of terrain is easily managed by using the option *Terrain>Edit Terrain* which opens the *Generation* window (see *Figure 4*), where the appearance of the terrain can be influenced by setting the parameters: feature size, noise, detail, variation, blurring and sharpness. These parameters determine the amount of land created the deformation of the surface, the random way of seeding of islands, the smoothing or sharpness of the surface.

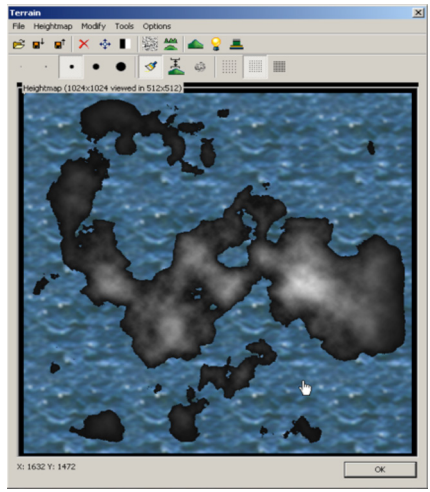

Figure 4; Creating and editing terrain.

The surface texture is easily generated with the option *File* > *Generate surface texture*. In the *Sandbox editor* more environment setup options are available for modifying terrain: vegetation tool, palette of effects as frozen level or weather effects. The created level can be exported and used in game mode. After creating the virtual world of the game the mission logic can be defined using the *Flow Graph* system in the *Sandbox* editor.

Flow Graph

The *Flow Graph* is a visual scripting system built into the *Sandbox Editor* for the *CryEngine2*. It is used by designers and programmers in creating environment interaction logic. The *Flow Graph* facilitates the process of game development by using visual scripting.

The *Flow Graph* uses nodes to represent entities or behaviors that can be controlled by linking them to other nodes. *Flow Graph* logic is stored in XML format and can be easily exported to disk, in order to be redistributed. A graph is always created and stored on a specific entity which has the benefit of the graph being exported with the object.

A *Graph* is the final result which defines a behavior; it consists of a series of *Nodes* which are *Linked* together via their *Port*.

A *node* is the representation of an entity or an action inside the *Flow Graph*. Node is a container of *Output* and *Input Ports.* There are two categories of nodes: *entity nodes* and *component nodes*.

Entity nodes represent entities, the input and output ports depend on the ports defined in the entity. Component nodes are nodes which can perform special functions but are not related to any entity.

A *node* consists of two sides, an input and an output side, the information transfer of the nodes is handled through *Ports. Ports* can send out or receive information. On the left side of a node are input ports used to connect incoming links, links from other nodes are connected into these ports. The ports on the right side of the node are called output ports and are activated depending on the behavior of the node. Ports are represented in the interface as small arrows on both sides of the nodes. Ports can have different data types; the data type is determinate with specific colors. A port can have one of six different types: any, *Boolean, Integer, Float, String, Vec3* (data type consisting of three float values, used to store positions, angles or color values).

Links are used to connect ports and transfer information between them. A *Link* connects an *Output Port* to an *Input Port* between two *Nodes.* Link is visualized as line between the ports of connected nodes.

Flowgraph Plugin System

The modding community provides a number of tools which reduce the labor of making new mods. A big percentage of the modding is based on flow graphs. In order to facilitate the modding process, the *Flowgraph Plugin System* was developed (Crytek, 2011).

The *Flowgraph Plugin System* aims to relive the process of distribution of new custom *Flow Graph* nodes. Previously, a custom mod dll that contained all the new nodes had to be built and this dll had to be distributed to each community member who wished to use the new nodes. The new defined *Flow Graph* nodes are separated into their own lightweight dll files – *plugin*; these *plugins* are detected and loaded by the system automatically merging its contents of *Flow Graph* system into the main system. To use any plugin, the *Flowgraph Plugin System* has to be installed in the relevant *CrysisWars* mod. The *Flowgraph Plugin System* is installed by extracting the Plugin DLLs into *FGPlugin's bin32* and *bin64* folders according to the dll versions. The users have to copy the dlls containing the *Flow Graph* nodes into that folder in the mod directory. To launch the mod in the *Sandbox Editor* the name of the mod has to be included in the path, with preceding –mod (see *Figure 5*). With lunching *Sandbox Editor* the included nodes will appear in the *Flow Graph Editor.*

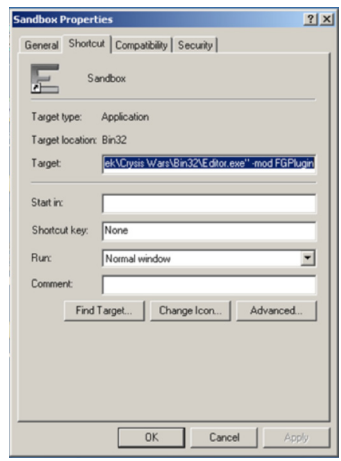

Figure 5; Including a mod in *Sandbox Editor.*

Setting up a programming environment

When the game *Crysis Wars* is installed onto the host pc, the *Software Development Kit Crysis Mod SDK* folder containing the necessary code is automatically created in *C:\Program Files\Electronic Arts\Crytek\CrysisWars\Mods*. The solution file is located in the folder Code named *GameDll.sln*. Before changing the code it is recommended to copy and rename the *CrysisWarsMod* folder.

In the newly created folder you can find and open *GameDll.sln* solution file in *Visual Studio*, we are using *Visual C++ 2005 Express Edition*.

To set up the properties of the project, right click on the project and *Properties* (see *Figure 6)*, set *Configuration to Active(Debug*), expand C/C++ section and in *Code Generation* change *Runtime Library* from *Multi-threaded Debug DLL(/MDd*) to *Multi-threaded)/MT*) and then click *Apply*.

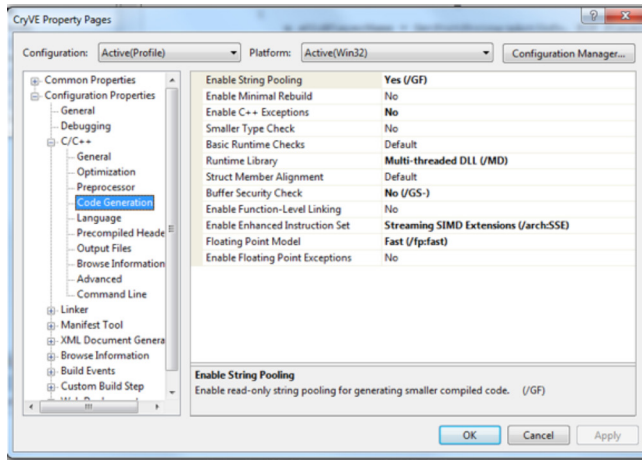

Figure 6; Setting properties for Code Generation.

The path of the output file can be set using the options in *Linker->General (see Figure 7)*

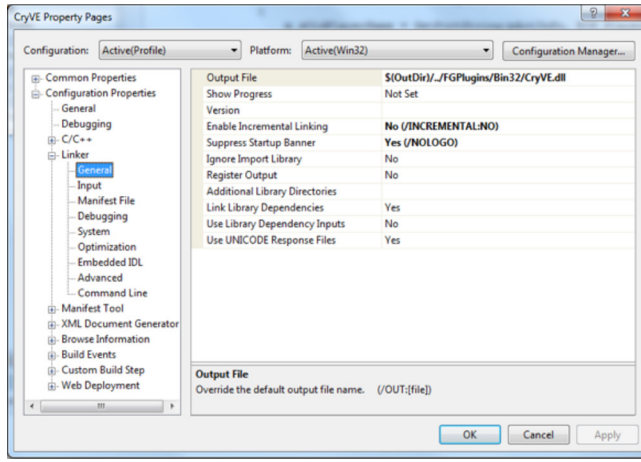

Figure 7; Setting General properties.

Creating a Custom Flow Node

The *CryVE* software uses game modification which aligns the camera view of specific instance of a game as a server or client in the *CryVE* setup. This implementation is done by creating a custom flow node which later will be connected in *Flow Graph*.

The custom flow node (*CFlowNode_CryVE*) created for the *CryVE* software has defined specific *Input* and *Output* ports. They are as follows:

Input Ports:

- new config is initialized (*EIP_SetNewConfig*),
- parameter for the name of the config file (*EIP_ConfigFileName),*
- input when we want to restrict the moving of the camera in different directions (*EIP_FixedCamMode)*,
- input when we want to fix the camera orientation (*EIP_FixCurrentCameraOrientation*).

Output Ports: message for multiplayer setup, used as debug message (*EOP_Message).*

Declaring the Class

After opening the solution file *GameDll.sln* in *Microsoft Visual Studio,* a new source file *CryVE.cpp* should be created in the *Nodes* filter of the *GameDll* project. More header files need to be included:

```
#include "StdAfx.h"

#include "GameRules.h"

#include "GameCVars.h"

#include "Player.h"

#include "Nodes/G2FlowBaseNode.h"

#include <string>

#include <fstream>

#include <iostream>
```

The precompiled header file and the *Nodes/G2FlowBaseNode.h* file define a class *CFlowBaseNode* which aids in creating a custom flow node.

The defined class *CFlowNode_CryVE* inherits the *CFlowBaseNode* class and overloads a few member functions to handle the logic behind the node. We will define the following member functions:

Constructor

Signature: Constructor(SActivationInfo pActInfo)*

The Constructor will need to take in a *SActivationInfo** as its only argument. The constructor is usually used for handling the member variables that have been defined.

Destructor

Signature: virtual Destructor(void)

GetConfiguration

Signature: void GetConfiguration(SFLowNodeConfig

&config)

This member function is called by the *Flow Graph System* to get the information about the node from the *Flow Graph Editor.* The argument config is an out variable and is used to supply the info about the node for the system.

ProcessEvent

*Signature: void ProcessEvent(EFlowEvent event, SActivationInfo *pActInfo)*

This member function will be called whenever an event needs to be handled and implements the functions of the node. Such events include its initialization and when a port becomes active or a key is pressed, what means we have to load different configuration. The event argument gives information about the event and *pActInfo* pass the information needed to handle the event.

Setting up the Configuration

Configuration of a node includes information about what *Input* and *Output* ports it has, the default values for any *Input Ports, Help strings* for the node and the ports, the default values for any *Input Ports* used to hold data.

GetConfiguration routine handles the default values and help strings. *SFlowNodeConfig* is one argument of *GetConfiguration* routine; this object has two member variables that allow specifying the input and output ports. These member variables are pointers to a data type *SInputPortConfig* and *SOutputPortConfig*. The ports are defined into two separate arrays, one for input and one for output ports and then supplied to the config object.

The *Ports* are defined in array format and each port has own index value in the array, the *portIDs* are defined in enumeration object and have to be in the same order. Two enumeration objects *EInputPorts* and *EOutputPorts* are declared to hold the index values of the ports. Using the ports defined from the beginning, the enumeration is:

Examples:

```
enum INPUTS {
  EIP_SetNewConfig = 0,
  EIP_ConfigFileName,
  EIP_FixedCamMode,
```

```
EIP_FixCurrentCameraOrientation
};
enum OUTPUTS
{
EOP_Message
};
```

The signatures of the template *InoutPortConfig* and *OutputPortConfig* functions are as followed:

Examples:

InputPortConfig<type> (Name, HelpString, HumanName, UIConfig)

InputPortConfig<type> (Name, DefaultValue, HelpString, HumanName, UIConfig)

InputPortConfig_Void (Name, HelpString, HumanName, UIConfig)

The *Port* arrays in *GetConfiguration* routine have to be static. Required arguments when defining a port are *Name* and *Default Value*. Help string is not mandatory but it is recommended, it helps when working with the node in the *Flow Graph Editor*.

Default Value argument is used to assign a default value to the port. This argument is used by setting the port to a common value which helps the user in the *Flow Graph Editor*. If this argument holds a string value, then all the next arguments have to be defined to at least NULL value otherwise the compiler may confuse the *Default Value* argument with *Help String* argument or it can throw an ambiguity error.

HelpString argument is used to define the help message the user will get in the *Flow Graph Editor* when hover the mouse over the port. This argument is not required but it is recommended to use a description which will explain the purpose of the port.

Example: _HELP("Call to do something!")

HumanName argument is used to specify a more human like name for the port used in the *Flow Graph Editor*.

Example: _HELP("My Port")

With the argument *UIConfig* an enumeration of possible values can be specified and the user can select one of those values when setting the value of the port. Enumeration is used when set of choices is limited and presented as a

list instead of free value.

Example: _UICONFIG(“enum_int:On=1,Off=0”)

Category of the node can be defined with using the function config. *SetCategory()* to filter the nodes which can be used in the *FlowGraph Editor*. The valid categories which can be set and they generally stand for: *EFLN_APPROVED, EFLN_ADVANCED, EFLN_DEBUG, EFLN_WIP, EFLN_LEGACY, EFLN_NOCATEGORY.* Depending on the set category the node can be approved and guaranteed to work*(EFLN_APPROVED)*, labeled as advanced*(EFLN_ADVANCED)*, used only for debugging purposes*(EFLN_DEBUG),* considered a Work in progress(EFLN_WIP), outdated and will be deleted *(EFLN_LEGACY)*, or it can be labeled as no category *(EFLN_NOCATEGORY).*

The *GetConfiguration* routine will be defined as:

```
void GetConfiguration( SFlowNodeConfig& config )

        {

                static const SInputPortConfig inputs[] =

                {

                        InputPortConfig<bool>(“SetNewConfig”,false,_HELP(“New configuration”)),

                        InputPortConfig<string>(“ConfigFilename”,_HELP(“Name config. file.”)),

                        InputPortConfig<bool>(“FixedCameraMode”, _HELP(“Fixed camera.”)),

                        InputPortConfig<bool>(“FixCurrentCameraOrientation”, _HELP(“”)),

                        {0}

                };

                static const SOutputPortConfig outputs[] =

                {

                        OutputPortConfig<string>(“Message”, _HELP(“Debug message”)),

                        {0}

                };
```

```
        config.pInputPorts = inputs;

        config.pOutputPorts = outputs;

        config.sDescription = _HELP("FG node that sets up a CAVE environment");

        config.SetCategory(EFLN_APPROVED);

    }
```

Handling the Events

ProcessEvent member function defines the logic for handling the events of the node. This routine has an event argument of type *EFlowEvent.* The datatype EFlowEvent is an enumeration with events most important arguments are:

- *eFE_Update* event, called when the node is updated.
- *eFE_Activate* event, called when one or more Input Ports are active.
- *eFE_FinalActivate,* has the same function as eFE_ Activate but is called after eFE_Update.
- *eFE_Initialize* event, called after the level has been loaded, then some basic initialization can be done.

The event argument can be wrapped in a switch statement and the events can be handled in case statements. *IsPortActive* is a helper function which gives information if the referred port is active.

In the CryVE setup we want to create a node which - depending on the configuration - will decide if a specific instance of a game is a server or client. Then the system is calibrated accordingly; if the current instance is a master, the mod sends a signal to potential clients (other slaves) that it is available to connect. If the instance is a client, the mod obtains the gaze of the master and aligns the camera view of the client accordingly. Once the cameras are aligned, CryVE reads a configuration file for the required image transformation (place translation, rotation and frustum shape transformation in 3D space).

In order to calibrate the system appropriately, there are some parameters that must be calculated depending on the geometry of the CAVE and the desired viewing position inside it. These parameters are the field of view (FOV), yaw,

pitch, and roll of the projection. M. Penna (M. Penna, 1991) showed that the yaw, pitch and roll parameters for the perspective projection of a quadrilateral can be calculated using

$$\alpha = tan^{-1}\left(\frac{r_{12}}{r_{11}}\right)$$

$$\beta = tan^{-1}\left(-\frac{r_{13}}{(r_{11}^2 + r_{12}^2)^{\frac{1}{2}}}\right)$$

$$\gamma = tan^{-1}\left(\frac{r_{23}}{r_{33}}\right)$$

where r_x are the components of a 3 X 3 matrix that defines the desired rigid motion rotation of the projection plane.

As the prototype implementation is a cubic CAVE, the desired point of view was fixed at the center of the cube. This simplifies the calculation of the camera look view parameters, resulting in

$$\alpha = \frac{\pi}{2}$$

$$\beta = 0$$

$$\gamma = 0$$

The calculation of the vertical and horizontal FOV is done by applying the formula, where H and W are height and width of the cube, respectively

$$FOV_{vertical} = 2\theta = \left(tan^{-1}\left(\frac{H}{2p}\right)\right)$$

$$FOV_{horizontal} = 2\varphi = 2\left(tan^{-1}\left(\frac{W}{2p}\right)\right)$$

However, given the cubic shape of the CAVE, we know that $FOV_{\text{vertical}} = FOV_{\text{horisontal}}$. Furthermore, we observe that p = W/2 = H/2, therefore, the formula for FOV (both horizontal and vertical) can be simplified to

$$FOV = 2\left(tan^{-1}\left(\frac{H}{2p}\right)\right) = 2\left(tan^{-1}\left(\frac{W}{2p}\right)\right)$$

$$FOV = 2(tan^{-1}(1))$$

$$FOV = \pi/2$$

The calculation of the parameters for each face of the cube is done in a similar way. After this, the process of translation and rotation is applied to each image frame before it is rendered by the clients. Figure 8 shows the process diagram for the CryVE mod.

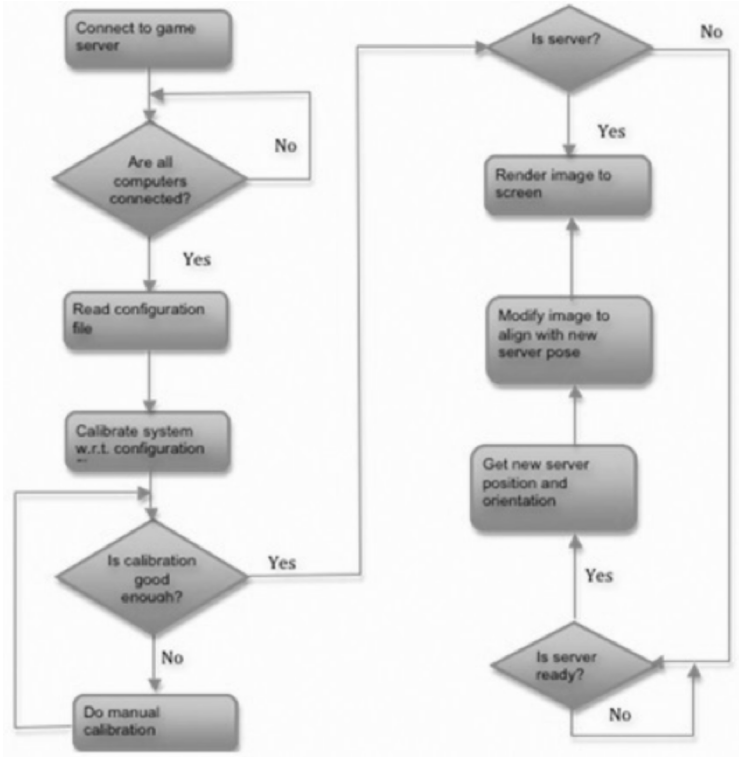

Figure 8; CryVE mod process diagram.

Registering with the Flow Graph System

The created custom Flow Node has to be registered to be used in the Flow *Graph Editor.* The system handles the node class according to which register macro is chosen. We are using *REGISTER_FLOW_NODE* macro with this registration, a new instance of the node is created each time it is used in a graph.

REGISTER_FLOW_NODE("Multiplayer:CryVESetup", CFlowNode_CryVE);

Other register macros which can be used to register the node

- *REGISTER_FLOW_NODE_SINGLETON,* uses a single instance of the class for all occurrences of the node regardless of where it is used.

- *REGISTER_FLOW_NODE_EX, REGISTER_FLOW_NODE_SINGLETON_EX* are extended registration methods and should be used if the node class is a template.

Using the Node in the Editor

According to registration of the node above, we can find it in the *Flow Graph Editor* under *Multiplayer/SetupCryVE*.

The *CryVE Flow Graph* can encapsulate the libraries produced by the mod into a component that can be reused in any *CryEngine 2* game. *The Flow Graph* defines input and output ports for the mod and connects them to other in game components, such as *player_HUD (Head-Up Display), player_id, player_position, and player_stance. Figure 9* shows the resulting *Flow Graph* in the *CryVE* plugin, along with other components and *Flow Graphs* already available in the game engine.

Figure 9; CryVE prototype *Flow Graph*.

Every *Flow Graph* must be encapsulated in a map for it to be used within a game. In a previous academic publication (Nakevska et al., 2011) we presented several case studies that ran as a mixed reality environment. For the needs of every application, a map is created, which also contains the *Flow Graph* and mod implementations. The map has to be installed in all the computers in the network.

A Virtual Museum Tour

We developed a conceptual design of a Virtual Museum Exhibition that show-cased historic events from the Netherlands (Juarez et al., 2009). The virtual environment resembled a medieval settlement, a virtual world that recreated sixteenth century Holland (see *Figure 10*). For this, a medieval game map was modified by adding dynamic interactions that could be selected by the visitors, leading to the main historic event. Visitors of the museum can use a portable controller in combination with CryVE to have an immersive experience. Using a handheld device, the visitor walks through the landscape and meets people (avatars) with whom he or she can interact. The handheld device is used as interface for interaction and in a subtle way it will introduce the visitor to the historic event that is about to happen in the virtual world. Other museum visitors can also join the exhibition. Each visitor has their own audio, and is able to select their own interaction with people in the virtual world through the handheld device. Having separate audio and shared visuals in the virtual world, allows visitors to have both individual and group experiences, and at the same time, it enables them to interact with people and objects in the virtual environment.

Figure 10; A tour in the Virtual Museum.

Virtual Garden with tangible interfaces

The Virtual Garden project is an attempt at aiding a layperson to design his or her garden, without requiring the user to be proficient with computers. A CryEngine-driven CAVE is used to create a convincing virtual environment. The user can explore and manipulate this environment through a tangible interface, which is basically a miniature representation of the virtual environment (see *Figure 11*). Because of this direct mapping between objects of the interface and of the visualization, users of the installation can understand the interaction and start to explore the virtual environment instantly. The current prototype is implemented using existing visualization

and VR techniques. The physical objects of the interface are fitted with pattern markers and are tracked by a webcam from underneath the objects. The position and rotation of the object from the video stream are written to an XML file where each of the objects is a child with three attributes (xpos, ypos, rotation). This XML file is subsequently read out by the game engine, which dynamically updates the positions of the objects in the virtual environment according to the acquired position data using a custom handler.

Figure 11; Impression of the Virtual Garden.

Conclusion

We presented *CryVE, CryEngine 2* game modification software which creates the new *Flow Graph* node needed to enable the *CryEngine* to be used as underlying software for a CAVE system. We presented the tools and methods needed for development of this kind of modification and the advantages of using game engine as an alternative in building virtual and mixed reality environments. With CryVE system we have created a platform for easily development of virtual environments; our challenge in the near future is to develop platforms which will easily integrate different inputs from mixed

reality environments. Problems which have to be solved are synchronization of the frame buffers, integration of different input devices and flexibility in incorporation of different sensors, actuators and input/output devices.

References

Barco Solutions (2011). i-space cave system: Multi-walled stereoscopic environment. Retrieved from *http://www.barco.com/en/product/732*

Cruz-Neira, C., Sandin, D., DeFanti, T., Kenyon, R., & Hart, J. (1992). The CAVE: audio visual experience automatic virtual environment. *SIGGRAPH*, June 1992.

Cruz-Neira, C., Bierbaum, A., Hartling, P., Just, C., & Meinert, K. (2005). VR juggler-an open source platform for virtual reality applications. *40th AIAA Aerospace Sciences Meeting and Exhibit.*.

Crytek (2011). Cryengine2 Specifications. Retrieved from http://crytek.com/cryengine/cryengine2/overview.

Flowgraph Plugin System for Crysis-Wars (2011). Retrieved from http://fgps.sourceforge.net/Help/main.html

HoloVis, Cave solutions (2011). Apollo CAVE solutions: Saving time and money through immersive visualizations. Retrieved from http://www.holovis.com/pdf/HoloVisCAVE.pdf

Jacobson, J. & Lewis, M. (2005). Game Engine Virtual Reality with CaveUT. *IEEE Computer 38 (4)*, 79–82.

Jacobson, J., Le Renard, M., Lugrin, J., & Cavazza, M. (2005). The CaveUT system: immersive entertainment based on a game engine. *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology,* ACM, 187.

Juarez, A., Schonenberg, W., Bartneck, C. (2011). LIVE HISTORY - a vision for the National Historic Museum in Arnhem, Netherlands. Retrieved from http://nhm.id.tue.nl.

Juarez, A., Schonenberg, B., & Bartneck, C. (2010). Implementing a Low-Cost CAVE system using the CryEngine2. *Journal of Entertainment Computing,* Vol1, Issue 3-4. Elsevier, Dec. 2010, 157-164.

Nakevska, M., Vos, C., Juarez, A., Hu, J., Langereis, G. & Rauterberg, M.

(2011). Using Game engines in mixed reality installations. *10th International Conference on Entertainment Computing,* Vancouver, Canada, 2011.

Penna, M. (1991). Determining camera parameters from the perspective projection of a quadrilateral. *Pattern Recognition,* 24 (6), 533–541.

Leigh, J., Dawe, G., Talandis, J., He, E., Venkataraman, S., Ge, J., Sandin, D. & DeFanti, T. (2001). Agave: Access grid augmented virtual environment. Proc. *AccessGrid Retreat,* Argonne, Illinois.

Sauter, P. M. (2003), Vr2go: a new method for virtual reality development. *SIGGRAPH Comput. Graph. 37* (1) (2003) 19–24. Retrieved from doi:http://doi.acm.org/10.1145/763993.763995.

Schou, T., Gardner, H. (2007). A Wii remote, a game engine, five sensor bars and a virtual reality theatre. *Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces, ACM,* 234.

Seeley, H. (2007). Game technology 2007: Cryengine2. *ACM SIGGRAPH 2007 Computer Animation Festival,* ACM, 64.

Unreal technology (2010). Retrieved from http://www.unrealengine.com/features

Are games worthy of academic attention? Can they be used effectively in the classroom, in the research laboratory, as an innovative design tool, as a persuasive political weapon? Game Mods: Design Theory and Criticism aims to answer these and more questions. It features chapters by authors chosen from around the world, representing fields as diverse as architecture, ethnography, puppetry, cultural studies, music education, interaction design and industrial design. How can we design, play with and reflect on the contribution of game mods, related tools and techniques, to both game studies and to society as a whole?

Contributors include:

Erik Champion, Peter Christiansen, Kevin R. Conway, Eric Fassbender, Jun Hu, Alex Juarez, Friedrich Kirschner, Marija Nakevska, Natalie Underberg

http://etc.cmu.edu/etcpress

Game Mods: Design, Theory and Criticism

Champion

Game Mods:

Design, Theory and Criticism

edited by Erik Champion

Game Mods: Design, Theory and Criticism

ISBN: 978-1-300-54061-8
Library of Congress Control Number: 2012956042

Game Mods:

Design, Theory and Criticism

edited by Erik Champion

Table of Contents