DirectX/C++ 기반 게임 소프트웨어의 공통 모듈 모형화 기법에 관한 연구※

변정원, 류성열
숭실대학교 컴퓨터학과
{Jimi01, syrhew}@ssu.ac.kr

A Study on Common Module Modeling Method of Game Software based on DirectX/C++

Jung-Won Byun, Sung-Yul Rhew
Dept. of Computer, Soongsil University

요 약

DirectX를 사용하는 윈도우 게임 및 콘솔 게임은 전 세계 게임 시장의 20% 이상을 차지하고 있으며, 전 세계 시장의 75% 이상이 C++로 개발되고 있다.

본 연구는 DirectX와 C++ 기반으로 개발된 20개 프로젝트 코드의 공통성과 가변성을 분석하여 패턴화하고, 이를 공통 모듈화 하였으며, 이 모듈을 모형화 하였다. 사례 연구를 통해, 제안한 기법을 사용하여 DirectX와 C++ 기반으로 개발된 소프트웨어 코드의 생산성이 60% 이상 개선됨을 입증하였다.

ABSTRACT

The windows games and the console games that use DirectX occupied 20% or more of world game market and that are developed by C++ Language occupied 75% or more of the market.

This study proposed that commonality and variability for code of the 20 projects based on DirectX and C++ is analysed and patternized, that common modules are created, and that modules are modeled. As a result of applying a case study, it is verified that the proposed model reduces effort in game development.

Keyword : Common Module, Modeling Method, DirectX / C++ Game

접수일자 : 2008년 10월 20일
일차수정 : 2009년 02월 11일
심사완료 : 2009년 03월 02일
※ 이 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.

1. 개 요

게임시장은 점차 커지고 있으며, 많은 사람들이 게임 개발에 참여하고 있다. Microsoft에서 제공하는 라이브러리인 DX(DirectX)는 게임 개발에 사용되는 대표적인 라이브러리 중 하나이며, DX를 이용하여 Windows 게임 및 일부 콘솔 게임이 제작되고 있다. 또한 다양한 지원 라이브러리로 인해, 게임 개발에 사용하는 언어 중 76%가 C++ 이다[1]. 게임 개발은 [2]의 통계를 분석한 결과, 게임을 대체할 다른 것이 나오기 전까지는 지속적으로 개발이 발생할 것이다. 그러므로 업계는 이러한 요구를 충족시킬 수 있는 방법을 마련하여 고품질의 게임을 빠르게 개발하여야 할 것이다[2].

이러한 기법으로 재사용에 대한 관심이 증가하고 있다[1][2]. DX를 사용하여 게임을 개발할 때, 이러한 재사용에 방해가 되는 문제점이 다수 존재한다. 만약 재사용 가능한 공통 모듈을 제공한다면 빠른 개발이 가능할 것이다. 그러나 DX/C++를 이용한 게임 개발 환경에 적합한 공통 모듈에 대한 연구는 회사의 역량에 따라 다르며, 학계의 연구도 많이 미흡한 것이 사실이다.

본 연구의 목적은 DX 및 C++를 사용한 게임 개발 환경(DX/C++ 기반)에서 게임 개발자가 게임 개발시 재사용할 수 있는 공통 모듈을 도출하고 모형화 하는 기법에 대하여 연구이다. 본 연구는 DX/C++ 게임 개발 프레임워크 연구를 위한 기반이 될 것이다.

본 연구를 위해, DX에 대해 살펴보고, 문제를 해결에 사용된 디자인 패턴과 Generic Programming 중 적용한 기법에 대하여 설명한다. 그리고 개발에서 DX 내부 흐름 아키텍처를 기반으로, DX를 실제 사용하는 부분에 대한 흐름을 분석하고, 이를 기반으로 공통성과 가변성을 식별하였다. 그리고 식별한 공통성과 가변성 중에 하나를 선택하여 여러 가지 방법을 적용하고 제안 기법을 연구하였다. 이렇게 만들어진 제안 기법을 DX/C++ 게임 개발에 재사용할 수 있도록 모형화 하였으며, 마지막으로 기존의 개발 방법과 제안한 방법을 통해 만든 코드의 양을 비교하여 본 연구에서 제안한 방법의 적용 가능성과 공통 모듈을 사용하여 얻는 이득을 표현하였다.

2. 관련연구

2.1 Direct X

DX는 Windows 및 일부 콘솔 게임 개발에 주로 사용되는 라이브러리이다. DX를 이용하여 게임을 개발하려는 개발자는 DXSDK (DirectX Software Development Kit)를 사용하여 Windows 또는 일부 콘솔 게임을 제작할 수 있다[3]. DX는 높은 성능을 요구하는 게임 및 멀티미디어 애플리케이션을 개발하기 위한 하위 수준의 API 집합이다[3]. DX 기반의 게임을 개발할 때에, DX API를 사용하여 게임을 개발한다. 그러나 본 연구는 DX를 이용하여 게임을 개발하는 전반적인 사항에 대한 것을 다루기에는 그 범위 너무 큼으로, DX를 사용하는 부분에 대하여 재사용 가능한 모듈을 만드는 것에 대해 범위를 한정한다. 그러므로 공통성 및 가변성에 대한 도메인은 DX를 사용한 게임 개발이 된다.

DX API를 이용하여 게임 소프트웨어를 개발할 때에 다음과 같은 문제점이 발생할 수 있다. 첫째 DX 기반의 공식적인 프레임워크의 부재로 인해 게임 개발 난이도가 높다. 둘째, 개발자는 인터페이스 호출을 위해 DX 구성요소 사이의 관계를 알아야 한다. 셋째, 개발사마다 설계의 방법이 차이가 있을 수 있지만, 게임 로직 코드와 DX 인터페이스가 혼재되어 게임 로직에 대한 가독성을 떨어트린다.

2.2 Generic Programming

Generic Programming의 목적은 소스 코드 레벨에서의 재사용을 추구한 설계/구현이다[4]. 대표적인 Generic Programming 중 하나가 C++의 표준 라이브러리인 STL(Standard Template Library)이다. 이러한 Generic Programming은 단순하고

명확한 기법들과 강력한 확장 및 조합을 장점으로 하고 있다. 본 논문에서는 이러한 Generic Programming 기법 중 매개변수 타입과 템플릿의 부분 특화를 사용한다.

매개변수 타입 : 매개변수 타입은 <T>를 의미한다[4]. 이러한 <T>는 구현시 자료 타입(int, float 등등)에 작용한다. 이 방법을 이용하면 공통 모듈 개발 시점이 아닌 공통모듈 이용 시점에서 타입이 변경될 수 있다. 이 방법은 향후 DX 버전과 프로젝트에 따른 가변치의 타입의 차이에 적용될 것이다.

템플릿의 부분 특화 : 템플릿 부분 특화를 통해 템플릿으로 가능한 구체화(Realization)시, 구현이 다른 클래스에 대응하는 것이 가능하다[4]. 이 방법을 통해 매개변수 타입 T에 따라 클래스의 구현이 달라질 수 있다는 것이다. 이 방법은 향후 DX 버전에 따라 사용할 수 있는 인터페이스와 사용할 수 없는 인터페이스를 공통 모듈 사용자에게 구분하여 노출할 때 사용된다.

2.3 기존 게임 엔진

기존 게임 엔진의 경우, 본 논문과 유사하게 서로 다른 환경을 지원하는 Cross API를 제공하고 있다. 범위나 규모면에서는 기존 게임 엔진이 훨씬 더 크지만(Engine, Tools, Resource 등), 목적상으로 서로 다른 환경에서 사용할 수 있는 공통모듈을 제작하는 공통성이 있다. [13] 게임 엔진의 경우, 이러한 다른 환경에 적용할 수 있게 Middleware를 이용하여 통합을 이루었으며, API의 경우, 각 언어에 따라 다르지만, Wrapper를 이용하여 통일성을 유지하였다. 그러나 이러한 기존 게임 엔진의 경우, 만들어진 제품을 따라 게임을 제작하는 것을 중점으로 하고 있으며[14], 본 연구는 재활용 가능한 공통 모듈을 제작하는 것을 목적으로 하고 있어, 그 범위가 다소 다름을 알 수 있다.

3. 공통 모듈의 생성을 위한 문제점과 해결

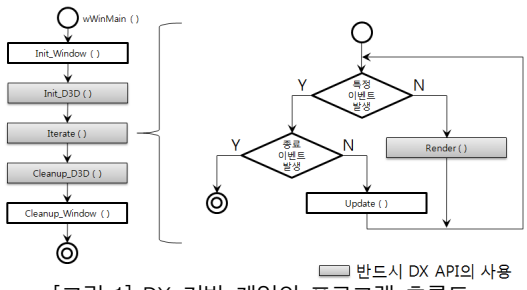
공통성과 가변성을 추출하기 위해, DX 내부 프로세스 관점에 아키텍처에 따라 DX Ver.9 버전의 6개의 프로젝트(DX Ver.9 에서 제공하는 Tutorial)와 DX Ver.10 버전의 14개의 프로젝트(DX Ver.10 에서 제공하는 Tutorial)를 분석하였다. 이러한 분석의 결과를 기반으로 DX 내부 프로세스의 관점과 API의 사이의 관계를 도출하였으며, DX 게임 프로젝트의 코드 흐름을 마련하였다. 이를 기반으로 DX 사용 부분에 대한 공통성과 가변성을 분석하였다. 분석 환경은 다음과 같다.

운영체제 : Windows Vista
개발 IDE : Visual Studio 2008
분석샘플 : DirectX Ver.9 Tutorial (6개),
DirectX Ver.10 Tutorial (14개)
소프트웨어의 구분 : Standalone, n-Tier의 클라이언트

3.1 공통성과 가변성

3.1.1 의미적 공통성

공통적인 부분은 프로그램 흐름에서 공통부분이 존재한다는 것이다. [그림 1]은 DX를 이용하여 게임을 개발할 때에 프로그램의 흐름을 표현한 것이다. [그림 1]에 따르면, 반드시 DX를 사용하는 부분은 크게 3부분으로 나뉠 수 있다.



[그림 1] DX 기반 게임의 프로그램 흐름도

Init_D3D : 프로그램 내에서 DX를 사용하기 위해 DX의 값을 설정하는 부분이다. 이 부분은 DX 내부 프로세스에 각 구성요소의 설정값들을 지정하는 부분이며, 고정된 Vertex, Index 데이터를 추가하고 텍스처를 지정하는 부분이다.

Render : 이 부분은 게임의 수행하면서 화면에 2D/3D 영상을 출력하는 부분이다. 필요에 따라서 동적으로 생성되고 변화되는 Vertex, Index 데이터나 텍스처 등을 생성하는 부분이 포함되기도 한다.

Cleanup_D3D : 프로그램의 종료 전에 프로젝트에 사용된 자원들을 해제하는 코드를 포함하고 있다.

[그림 1]의 각 부분은 DX Ver.9와 Ver.10 사이의 공통성으로 추출할 수 있다. 본 연구에서는 이들의 공통성을 의미적 공통성으로 정의한다.

3.1.2 가변성 분석

이렇게 추출한 공통성을 기반으로 가변성 분석하였다. 분석 절차는 [5] 연구의 절차를 따랐으나, 요구사항이 아닌 코드를 기반으로 분석을 수행하였다. 분석의 결과 가변점 타입은 Attribute/Interface이고, 가변 집합을 살펴보니, API의 차이와 인터페이스에 들어가는 파라미터의 차이로 분류할 수 있었다. 가변점의 집합의 수는 API의 경우 API의 개수가 되며, 파라미터의 경우 프로젝트에 따라 다른 설정값의 개수와 같음으로 개수가 Open임을 알 수 있었다.

가변성은 공통성을 기반으로 DX 버전별 코드를 대상으로 조사하였다. [표 1]은 DX Ver.9와 DX Ver.10의 디바이스 초기화 프로젝트의 LOC (Line of Code)를 비교한 것이다. DX Ver.9와 DX Ver.10의 코드의 내부보다는 전체적인 모습이 차이가 났다. [표 1]은 의미적으로 공통성을 가지는 부분으로 코드를 재구성하여 LOC 비교한 것이다. 실제 코드의 차이는 전체 144 LOC 중에 98 LOC가 차이가 있어 거의 68% 정도가 차이가 남을 알 수 있다. 결과적으로 가변성의 범주는 DX API의 차이와 설정값의 차이로 구분할 수 있다.

[표 1] DX Ver.9과 DX Ver.10 사이의 LOC 차이

구분	LOC		가변성 범주
	DX9	DX10	
윈도우 등록	25	35	설정값
DX 디바이스 초기화	25	38	DX API / 설정값
화면 출력	23	10	DX API / 설정값
DX 디바이스 해제	11	15	DX API

[표 2] 코드 영역에서의 공통성과 가변성의 세부 분할

공통성	세부 분할	가변치 범주
DX 디바이스 초기화	설정값 지정	설정값
	디바이스 초기화	DX API
화면출력	화면출력을 위한 설정값 지정 (장면)	설정값
	화면 출력	DX API

공통성을 단위로 각 버전마다 재사용 가능한 단위로 묶어서 제공한다면 재사용성이 향상될 것이다. 그러나 설정값에 의한 차이와 DX API의 차이가 혼재되어 있는 부분이 존재하기 때문에 공통성을 단위로 더 세부적으로 분할하였다. 분할 한 결과는 [표 2]와 같다.

3.2 DX 버전의 API 차이

DX 버전에 따른 차이를 해결하기 위해선, 차이가 나는 부분은 하나의 모듈로서 제공하는 것이다. 이러한 모듈의 단위는 공통성을 한 단위로 한다. 모듈로 제공하는 방법은 함수로서 제공하는 방법이 있다. 단순히 함수 단위로 모듈을 제공할 경우 실제 모듈을 사용하는 곳이 다수이기 때문에, 버전이 바뀌면 함수를 사용하는 곳에 대해 일일이 바꾸어주는 문제가 발생한다.

이러한 문제를 해결하기 위해, 모듈을 클래스 단위로 만들어 사용할 수 있다. 공통성에 대해 인터페이스를 만드는 방법도 있다. 그러나 함수는 통일이 되었을 지라도 버전이 바뀌면 클래스 명을 변경하기 때문에 결국은 함수 단위로 모듈

[표 3] 다른 클래스의 상속을 이용한 해결

공통적으로 제공	DX Ver.9	DX Ver.10

class DX_9
{
Initialize() {
// DX Ver.9의
// 디바이스
초기화
}
};
class DX_10
{
Initialize() {
// DX Ver.10의
// 디바이스
초기화
}
};
	// 차이점
	class DX
	: DX_9 {};
		// 차이점
		class DX
		: DX_10 {};

	// 사용 부분
	DX device;
	device.Initialize();	

을 제공하는 것과 차이가 없다. 이러한 차이를 해결하기 위해서 [표 3]과 같은 기법을 사용할 수도 있다. 이러한 경우 사용 부분이 변경되지 않음으로 버전에 따라 고치는 부분이 매우 줄어들게 된다. 이러한 방식으로 코드를 변경하면 코드의 전체적인 모습이 [표 4]와 같은 모습으로 변하게 된다. 결과적으로 DX API로 인한 문제는 모두 해결하고, 가변치의 범주는 설정값에 대한 문제만 남게 되었다.

[표 5] 다른 클래스 상속을 적용한 DX Ver.9과 DX Ver.10 사이의 LOC 차이

구분	LOC		가변성 범주
	DX 9	DX 10	
윈도우 등록	25	35	설정값
DX 디바이스 초기화	5	26	설정값
화면 출력	2	4	설정값

3.3 프로젝트 설정값의 차이

프로젝트에 따라서 설정값이 차이가 날 수 있다. 이러한 설정값의 차이는 [표 5]와 같이 두 가지 분류로 나뉠 수 있다. 하나는 설정값의 Value의 차이이며, 나머지 하나는 동일한 의미의 설정값의 타입의 차이이다.

[표 6] 설정값의 차이의 종류

비고	Project A	Project B
값	int a = 3;	int a = 9;
타입	int a = 3;	string a = "three";

DX API의 차이를 의미적인 공통성을 기반으로 해결하였다면, 설정값의 차이의 경우 문법적인 방법으로 해결하여야 할 것이다. 타입의 차이의 경우, 우선 생각할 수 있는 방법은 함수 오버로드이다. 그러나 재사용 모듈을 사용하는 곳에 차이가 있기 때문에 모듈의 재사용성이 떨어지게 된다. 이것을 해결하기 위해, 설정값을 설정하는 부분을 클래스 내에 포함하여 이를 상속받아 가변성을 구현하여 사용하는 방법도 있다. 이러한 방식의 경우, 재사용 모듈을 사용하여 게임을 개발하는 자에게 설정값과 관련된 가상함수를 설명해 주어야 하는 단점이 있다. 두 번째 문제는 DX 버전에 따라 설정값의 의미가 거의 유사하지만 타입이 다르다는 점이다. DX Ver.9와 DX Ver.10 사이에 이러한 차이는 설정값의 구조체를 표준화 및 단일화하는 것을 어렵게 만들 수 있다. 의미적으로 같은 값을 가지나, 실제 다른 구

[표 4] 템플릿 부분 특화를 적용하여 가변성을 해결한 결과

공통적으로 제공	DX Ver.9	DX Ver.10

// 외부파일에서 설정값 로드
template <> class Load_Value <9> {
public: static int ret_param() {
return L_V_int(); }
};
template <> class Load_Value <10> {
public: static string ret_param() {
return L_V_string(); }
};
	class User_DX_Initialize
	: DX_Initialize <9>
	{}
		class User_DX_Initialize
		: DX_Initialize <10>
		{}

	// 사용 부분, 변동 없음
	User_DX_Initialize device;
	device.Initialize();	

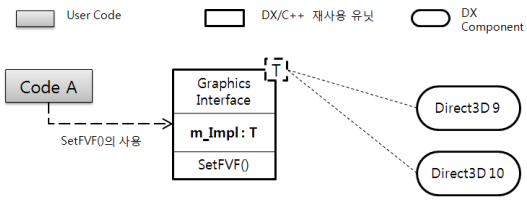
조체의 타입으로 인해 재사용성에 문제가 발생할 수도 있다.

값의 차이의 경우, 하드코딩 방식은 문제가 될 수 있다. 이러한 하드코딩을 피하기 위해선, 현재 사용되고 있는 방식은 값을 외부파일로 두어 호출하는 방식을 주로 사용하고 있다. 외부파일의 원하는 값의 의미를 부여하기 위해 XML을 사용하고 이를 파싱하는 방식도 사용되고 있다. 파싱 코드의 경우 본 연구의 범위가 아님으로 본 연구에서 추가적인 설명을 하진 않을 것이다.

설정값의 차이에 의해 발생한 문제를 해결한 후, 프로젝트에 따라 다음과 같은 유사 패턴을 발견할 수 있었다. ret_param()의 결과 값만 다르며, 둘째, L_V_int(), L_V_string()만 다른 것이다. 이러한 공통점도 해결이 된다면, 결국 이 모듈을 사용하는 자는 L_V_int()인지, L_V_string()인지만 결정하면 되는 것이다. 이러한 공통점을 해결하기 위해 다중 상속을 사용할 수 있다고 판단했다. 이러한 방식으로 수정하였지만 결과적으로는 컴파일이되지 않았다. 이유는 L_V_string과 L_V_int()의 반환값이 서로 충돌 나는 현상이 발생하였다. 이러한 문제는 Generic Programming의 템플릿 부분 특화를 사용하여 해결할 수 있다. 결과는 [표 6]과 같다.

4. 제안 기법의 모형화

제안 방법을 모형화하여 일반적으로 DX에 적용할 공통 모듈을 모형화할 것이다. 이 장에서 사용된 방법의 개념적인 클래스 다이어그램으로 표현하여 실제 DX 공통 모듈을 이용하는 API의 차이와 설정값의 차이 부분에 모두 적용될 수 있다. 공통 모듈은 앞서 말한 방식을 통해 재사용 가능한 코드 부분을 도출하여 공통 모듈을 제작하고 가변적인 부분에 대해서는 부분 특화를 사용하였다.



[그림 2] DX 버전에 대한 독립성 지원 방법
(Bridge + 매개변수화된 타입)

4.1 상위 API를 통한 DX API 독립성 지원

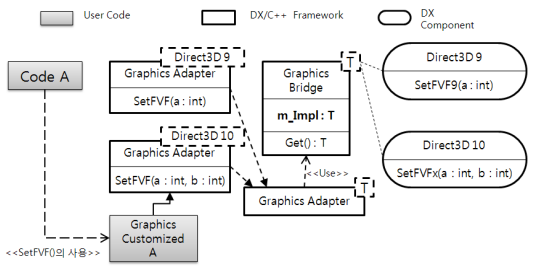
DX 버전에 따른 API 변경 문제를 해결하기 위해, Bridge 패턴[6]을 적용할 수 있다. Bridge 패턴을 적용한 기법은 [그림 2]와 같다. 다른 DX 버전에 따라 다른 구현 클래스(Direct3D 9, Direct3D 10)의 공통 기능(EX : 디바이스의 초기화)에 대해, 부모 클래스(Graphics Implement)를 만들고, 부모 클래스를 소유하는 연결 클래스(Graphics Bridge)를 통해 DX에서 제공하는 기능을 상위 API로 사용자가 사용하는 것이다. 이러한 Bridge 패턴을 통해 사용자는 DX 버전과 관계없이 구현이 가능하게 된다.

Bridge 패턴을 사용할 경우, DX가 상위 부모를 재공하지 않는 한, DX 버전에 대한 부모 클래스를 제작하기 힘들다. 그러므로 본 논문에서는 Bridge 패턴의 핵심(포함과 간접사용)을 표현할 수 있는 매개변수화된 타입(Parameterized Type) [BJA97]을 사용할 것을 제안한다. 매개변수화된 타입을 사용하면 연결 클래스에 타입과 상관없이 다른 클래스를 직접 포함 및 사용할 수 있다. 따라서 여러 가지 구현 클래스에 대한 부모 클래스가 없이 직접 포함할 수 있게 된다. 이러한 직접 포함은 매개변수(T)를 통해 가능하다. [그림 2]는 매개변수를 사용한 방식을 표현한 것이다. 매개변수화된 타입을 통해 얻을 수 있는 장점은 첫째, 환경을 설정하는데 있어 T의 변경만으로 공통적인 API를 사용할 수 있다. 둘째, 재사용 유닛을 사용하는 사람은 공통적인 API를 제공받을 수 있다.

4.2 프로젝트 설정값과 구현의 분리

프로젝트마다 다를 수 있는 DX 설정값을 조정하기 위한 기존의 방식은 설정값을 입력할 수 있는 함수를 제공하는 것이다. 그러나 설정값 조정 함수는 많이 사용되진 않지만, 함수는 계속 노출되는 단점을 가지게 된다.

이러한 가변성을 해결할 수 있는 디자인 패턴은 Adapter 패턴[6]있다. 설정값을 가지는 설정



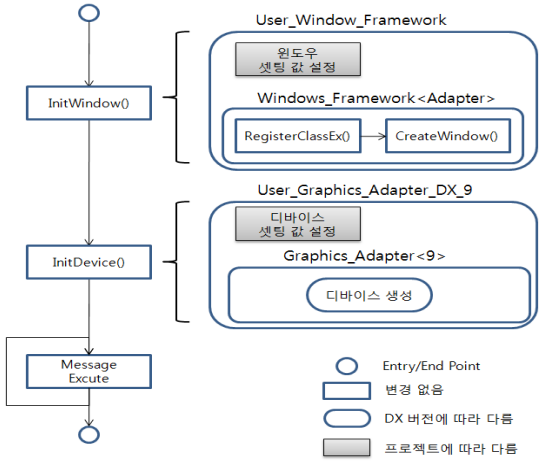
[그림 3] 가변성 지원 방법
(Adapter + 부분 특화)

클래스를 중간 클래스를 두고, 이를 상속 사용자에 맞게 특화함으로서 해결할 수 있다. 이러한 Adapter 패턴은 목적에 따라 API 변경에 대해서도 사용할 수 있다. 그러므로 DX 버전 독립성 지원에서 제시한 기법과 같이 사용할 수 있다. Adapter 패턴의 적용은 변경될 부분을 다루는 중간 클래스를 통해 변경이 발생한 경우, 중간 클래스만 바꾸는 국지적 변경의 이득을 얻을 수 있다. 그러나 Adapter 패턴은 중간 클래스가 다수 버전에 대한 인터페이스를 가져야 하는 문제점을 가진다. 그러므로 DX Ver.9 버전을 이용하는 사람도 제공되지 않는 DX Ver.10 버전의 인터페이스에 접근할 수 있는 문제점이 발생한다. 이러한 문제점을 해결하기 위해 템플릿 부분 특화 기법[4]을 사용할 수 있다. 부분 특화 기법은 같은 클래스에 매개변수를 다르게 하여 구현을 다르게 할 수 있다. 물론 DX 설정값에 대한 커스터마이즈 코드는 매개변수가 지정된 Adapter를 상속 받아서 특화하면 된다. [그림 3]은 이러한 부분 특화 기법을 이용한 Adapter를 적용한 방법이다.

5. 적용과 평가

본 연구에서 제안한 기법에 대하여, DX에 가장 기초가 되는 디바이스의 초기화를 적용한 사례이다. [3]의 디바이스 초기화를 기존 방법을 사용한 사례 [그림 1]과 본 연구에서 제안한 기법을 적용한 사례 [그림 4]를 비교할 것이다.

[그림 1]에 따르면, Init_Window나 Init_D3D의 경우 DX와 프로젝트에 따른 차이로 인해 재사용이 불가능하였다. 프로젝트에 따라 다른 코드 영역을 재사용 가능한 코드 영역과 분리하고, 제안한 기법을 적용하였다. 이러한 기법을 적용하여,



[그림 4] 가변성 지원 기법의 적용

DX 버전과 프로젝트의 차이에 무관하게 재사용 가능한 부분을 추출하여 독립성 지원 기법을 적용할 수 있다. 재사용 가능한 부분을 추출한 후, 프로젝트에 따라 다른 부분은 상속을 통해 커스터마이즈된 유닛으로 만들 수 있다. 적용사례는 [그림 4]와 같다.

제안한 기법의 실제 코드 영역은 [표 7]과 같다. 이러한 기법의 핵심은 변할 수 있는 부분과 변하지 않는 부분의 분리이다. 이러한 분리를 통해 프로젝트와 DX 버전에 무관한 재사용 가능한 모듈을 생성할 수 있었다.

이러한 기법을 적용한 결과는 동일한 DX 초기화라는 프로젝트를 수행하기 위한 작성 코드량

[표 7] 제안한 기법의 코드 영역

DX Ver.9	class User_DX : public DX<9>
	{ };
	...
	// InitWindow
	User_Windows<User_DX> tFramework();
	// InitDevice
	User_DX tDevice();
	// Message Excute
	tFramework.Message_Excute(tDevice);

DX Ver.10	class User_DX : public DX<10>
	{ };
	...
	// InitWindow
	User_Windows<User_DX> tFramework();
	// InitDevice
	User_DX tDevice();
	// Message Excute
	tFramework.Message_Excute(tDevice);

(LOC)이 줄어든 결과를 가져왔다. 이에 대한 분석 결과는 [표 8]과 같다. 프로젝트 생성은 어떠한 코드도 없는 상태에서 동일한 결과를 만드는데 필요한 LOC이며, 버전의 변경은 DX Ver.9에서 DX Ver.10로 또는 그 반대의 경우 기존 코드에서 변경해야하는 LOC를 작성한 것이다.

다른 적용 사례는, 본 연구에서 제안한 기법을 이용하여 간단한 게임(Alcanoid, Pingpong)를 제작하였다. 기본적인 리소스는 동일하고 위의 제안 기법을 기반으로 DX Ver.9, DX Ver.10으로 제작하였다. 제작에 사용된 리소스가 동일하다고 가정할 때, 이를 기반으로 개발 노력은 [표 9]와 같다. 표에 따르면 전체적인 LOC는 기존 방법보다 제안된 기법을 사용할 때가 많다(1133 : 1418). 그러나 Pingpong 게임 제작시 기존 방법은 Alcanoid와 유사하나 각기 다른 점으로 인해 재사용율을 판단하기 힘들며, 또한 재사용율도 낮았다. 그러나 제안된 기법의 경우, 기존 공통 모듈에 추가 모듈로 인한 약 10%의 변동분(68 LOC)을 제외하면, 코드 제작량은 기존 방법에 비해 약 1/3(964 : 342)로 약 30%의 노력만이 듦을 알 수 있었다.

결론적으로 본 연구에서 제안한 기법을 이용하여 공통 모듈의 모형화 및 제작이 가능하며, 이를 활용하면 게임 개발시 투입된 노력을 줄일 수 있었다. 그러나 본 연구에서 제안한 방법이 완전한 방법은 아니며, 지속적으로 모듈 역시 보강되어 가는 단점이 있으며, 모듈의 크기 역시 점점 커지는 단점도 있었다. 실제 컴파일한 결과(오브젝트 코드)는 제안 기법을 사용한 DX Ver.10의 경우 기존 DX Ver.10과 비슷하지만, 제안 기법을 사용한 DX Ver.9의 경우 DX Ver.9와 비슷하지만, 공통 모듈의 공통성을 지원하는 코드 영역의 추가로 인해 전체적인 목적 코드량은 약간 증가한 결과를 가져왔다.

[표 8] 기존 방법과 제안된 방법을 적용한 개발 노력의 비교 – DX Version 및 프로젝트 설정

구분	기존 방법 (LOC)			제안된 기법 (LOC)		
	공통 모듈	DX Ver.9	DX Ver.10	공통 모듈	DX Ver.9	DX Ver.10
프로젝트 생성	-	168	241	340	14	27
버전 변경	-	112 삭제 184 추가	184 삭제 112 추가	-	11 삭제 17 추가	17 삭제 11 추가
		총 296 변경			총 28줄 변경	

[표 9] 기존 방법과 제안된 방법을 적용한 개발 노력의 비교 – 실제 게임 제작

구분	기존 방법 (LOC)			제안된 기법 (LOC)		
	공통 모듈	DX Ver.9	DX Ver.10	공통 모듈	DX Ver.9	DX Ver.10
Alcanoid	-	1133	1264	642	776	1051
Pingpong	-	964	1097	710	342	388

6. 결 론

본 연구는 DX 버전과 프로젝트 설정값에 따른 차이에 무관하게 재사용할 수 있는 공통 모듈의 개발 기법에 대하여 제안하고 모형화 하였다. 기법을 제안하기 위해, DX Ver.9와 DX Ver.10의 내부 프로세스를 통해 게임 개발 코드에서 DX API의 사용 부분을 도출하였으며, 20개의 DX 샘플 프로젝트를 분석하여 DX로 게임을 개발하는 프로그램 흐름을 도출하였다. 이러한 흐름의 의미적 공통성으로 식별하고, 이에 대해 가변성 분석을 하여 가변치의 타입과 가변치의 범위 및 가변값의 범주를 식별하였다. 이를 기반으로 공통 사용할 수 있는 모듈을 개발하기 위해 디자인 패턴과 Generic Programming의 일부 기법을 이용한 개발 기법을 제안하였으며, 이를 실제 적용한 사례를 제시하고, 기존 방법과 비교하여 재사용 가능한 공통 모듈 사용이 적은 개발 노력량이 듦을 검증하였다.

본 연구를 수행하면서 다음과 같은 한계점을 발견할 수 있었다. 첫째, 공통 모듈을 제공한다고 하더라도 기존 코드를 제안한 공통 모듈에 따라 작성할 것인가에 대한 질문이며, 둘째, 프로젝트에 따라 다양한 상황이 발생하는데 이러한 요소를 고려하지 않았다는 점이다. 셋째는 DX API에 대하여 전체적인 API의 대조를 수행하였지만, 실제 사용하는 부분에 대한 고려는 초기화하는 규모가 매우 적은 프로젝트(샘플 및 단순한 게임)에 적용해서, 이러한 것이 일반화되지 못했다는 점이다.

향후 연구는 본 연구를 기반으로 DX/C++ 기반 게임 개발 프레임워크를 마련하고 이를 기존 게임 개발 프레임워크와 비교하여야 한다. 그리고 실제 프로젝트에 적용하여 그 실효성을 검증하며, 개발 프로세스와 제안한 기법 사이의 관계를 마련하여야 한다.

참고문헌

[1] GITISS, “게임 백서 2007”, 한국게임산업진흥원, 2007

[2] GITISS, “게임 백서 2006”, 한국게임산업진흥원, 2006

[3] Microsoft, "The DirectX Software Development Kit", Micros oft Press, 2007

[4] Andrei A., "Modern C++ Design: Generic Programming and Design Patterns Applied", Pearson Education, 2001

[5] 장수호, 김수동, 공통성 및 가변성 분석을 활용한 컴포넌트 설계 기법, 정보과학회논문지 - 소프트웨어 및 응용, 제 31권, 제 6호, pp.716-727, 2004

[6] E. Gamma, R. Helm, R. Johnson, Jone V., "Design Patterns: Element of Reusable Object-Oriented Software", Addison Wesley, 1995

[7] Appleton B., "Patterns and Software: Essential Concepts and Terminology", http://www.cmcrossroads.com/bradapp/docs/patterns-intro.pdf, 1998

[8] Stroustrup Bjarne, "The C++ Programming Language, 3rd ed", Addison-Wesley, 1997

[9] Coplin James O., "Advanced C++ Programming Styles and Idioms", Addison-We sley, 1992

[10] Dirk Riehle, "Framework Design : A Role Modeling Approach", Institute of Computer Systems, 2000

[11] Meyers Scott, "More Effective C++", Addison-Wesley, 1995

[12] Mohamed F., Douglas S., Ralph J., "Building Application Frameworks: Object-Oriented Foundations of Framework Design", Wiley& Sons, 1999

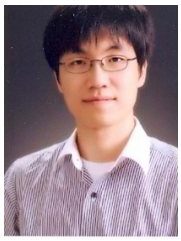
[13] "Implementing Gameplay Elements Using Gamebryo", Emergent Game Technologies, 2009

[14] 강희정, “아주 엔진 : 렌더링과 씬 모듈을 중심으로 한 3D 게임 엔진 아키텍처 연구, 아주대학교, 2004

[15] Andre Rollings, Dave Morris, "Game Architecture and Design", New Riders Games, 2003

[16] Mark Deloura et al., Game Programming Gems, 정보 문화사, 2001

[18] Mark Deloura et al., Game Programming Gems 3, 정보 문화사, 2003



변정원(Byun JungWon)

e-mail : Jimi01@ssu.ac.kr

2001년~2007년 숭실대학교 미디어학부 학사
2008년~현재 숭실대학교 컴퓨터학과 석사과정

관심분야 : 소프트웨어 아키텍처/프레임워크, 소프트웨어 요구공학, 게임 개발 프로세스

류성열(Rhew, SungYul)

e-mail : syrhew@ssu.ac.kr

1981년~현재 숭실대학교 교수
1982년~1995년 숭실대학교 전자계산연구소 및 중앙전자계산소 소장
1997년~1998년 George Mason University 객원 교수
1998년~2001년 숭실대학교 정보과학대학원 원장
2004년~현재 한국품질재단 운영위원회 위원장
2006년~현재 공정거래위원회 성과관리위원회위원
2008년~현재 정보통신연구진흥원 비상임 이사

관심분야 : 소프트웨어 유지보수, 게임 개발 프로세스, 오픈소프 소프트웨어