CMP
United Business Media

REVIEWS: GAMECODA, SKETCHBOOK PRO, MODEL BANK

FEBRUARY 2004

# Game Developer

ON THE FRONT LINE OF GAME INNOVATION

## Ready, Aim, Hire!

Our 3rd Annual Salary Survey

## Intrepid's B.C.: Taking Animation Approximation Beyond Lerp

## Inner Product: Designing A New Language

## Postmortem: Totally Games' Secret Weapons Over Normandy

$5.95US $6.95CAN
02
0 71486 02133 9
Display Until March 18, 2004

글 / 조나단 블로우

# 게임 기술 컨설턴트가 전하는 언어 러프(Lerp) 설계: 2부

두 달 전 필자는 게임 프로그래밍을 간소화 시키는 데 도움이 되는 방법으로 서술 논리를 고찰했다. 러프 이면의 패러다임은 전통적인 명령형 프로그래밍과 서술 논리의 선언적인 스타일간의 융합이다.

지난 달 기고문 후반에서 서술 논리 데이터베이스에 대한 통사적인 겉치레(Syntactic Sugar)로 C-스타일 구조체를 실행한 바 있다. 즉 이 프로그래밍 언어의 내성 지원이 매우 뛰어나며 근본 데이터 구조 위에서 모든 종류의 흥미로운 패턴 매칭 검색을 실행할 수 있다는 얘기다.

늘 무작위의 사람들이 높은 수준의 프로그래밍 기능들을 제시하고는 했는데, 처음에는 그럴듯하게 들리지만 막상 문제에 적용하면 실제로는 도움이 되지 않는 경우가 허다했다(사실 이것은 신규 언어 기능에서는 일반적인 것으로 보인다). 필자 역시 게임 업계에서 공인되지 않은 기능들을 제시했을 뿐 아니라 전통적인 서술 언어의 견인차인 구조체를 대체하는 데 그것을 사용하기도 했다. 이 모든 것을 감안하면 필자에게는 평범하지 않은 실제 문제에 유용한 데이터베이스 기능을 제시할 의무가 있다.

필자는 초기 컬럼을 통해 프로그램을 선택해 그것을 C++에서 러프로 옮기면 재미있겠다고 생각했다. '인터랙티브 프로파일링: 1부(2002년 12월)'에서 이 애플리케이션을 선택한 바 있다. 이 프로그램은 비교적 단순하다. 마우스 룩 키보드 워크 인터페이스로 움직이고 그라운드는 평평하며, 약 100여 개의 크레이트가 흩어져 있다. 이들 크레이트는 각기 다른 방향으로 회전하며 그 위에 다수의 각기 다른 텍스처가 있다. 원래 이 프로그램의 포인트는 프로파일러가 실행할 단순한 세계를 제공하는 것이었다. 프로파일링은 이 방법의 포인트가 아니었기 때문에, 필자는 포트를 단순화시키기 위해 전통적인 프레임 카운터까지 프로파일링 정보를 줄였다.

이 애플리케이션은 초기 언어 설계에 안성맞춤이었다. (서술 이론을 기반으로 하는 언어를 원한다는) 막연한 생각으로 시작해 이처럼 견고한 프로그램에 의한 요건을 토대로 선택적으로 언어를 다듬었다. 프로세스 처음부터 끝까지 현실에 발을 딛고 싶었던 것이다.

## 테스트 프로그램의 성격

이 테스트 프로그램의 흥미로운 점은 매우 낮은 수준에서 모든 크레이트를 이끌어 낸다는 것이다. glVertex(), glTexCoord(), 및 glColor()를 사용해 한 번에 하나의 마루점(vertex)에서 각각의 크레이트를 렌더링한다. 러프의 의도는 스크립팅 언어이며, 실제 게임에서는 이처럼 낮은 레벨에서 스크립팅 언어를 사용해 렌더링하는 경우가 거의 없다. render_mesh() 기능 대신 C++ 코드를 사용하는 것이 빠르고 효과적일 것이다.

사실 이 테스트 애플리케이션은 레벨이 너무 낮아서 표면 법선(surface normal)까지 동적으로 산출하고 그것을 빛의 방향으로 내적화(dot-producting)시킴으로써 표면 자체의 조명을 산출한다. 프로그램이 작업량을 줄이려 할 경우 각 개체의 표면 법선을 저장하고 OpenGL에게 조명 자체를 실행하라고 지시할 수 있다.

그 결과 이 프로그램은 실질적으로 우리의 도메인 요건에 필요한 것보다 훨씬 강조됐다. 빠르게 실행될 수만 있다면 좋은 것이다.

## 프로그램 근간

이 테스트 애플리케이션은 벡터3 클래스를 사용해 포인트를 표시하고 이들 포인트 상에서 일련의 연산을 수행한 후 그 결과를 앞서 언급한 Open GL에 전달해 렌더링을 명령한다. 자연히, OpenGL 기능을 러프로부터 호출하는 방법을 실행해야 했다. 방법은 간단하다. 러프로부터 호출할 수 있는 C++로 래퍼(wrapper)를 암호화하면 된다.

또한 래퍼를 통해 처리된 C++로 벡터3 클래스를 실행할 수도 있었지만 그것은 도피에 지나지 않았다. 이 프로그램의 전반적인 포인트는 (테스트 러프 데이터 구조의) 처리 메커니즘을 강조하는 것이며, 벡터3는 이 프로그램의 근본 데이터 구조이다. 따라서 벡터3는 이 데이터베이스 기능을 사용해 실행해야 하는 것이다.

조나단 블로우 | 조나단(jon@number-none.com)은 게임 기술과 연관된 사람들을 상대로 하는 게임 기술 컨설턴트이다.

## 벡터3 실행

벡터3를 하나의 데이터베이스로 실행하는 방법은 무엇일까? 각각 부동 소수점 값이 붙어있고 x, y, z로 이름붙인 슬롯에 지나지 않은 C++를 사용할 수 있었는데, 우리가 흔히 애용하던 접근법이다. 그러나 보다 높은 차원으로 매우 원활하게 일반화되지 않는다는 사실을 깨닫고는 이와 같이 이름이 붙여진 좌표를 재고하게 됐다. 4차원의 경우 물론 w를 사용할 수 있지만 그 이상에서는 이 문자가 임의적이 돼버린다. 또한 러프가 고차원 언어가 되면 벡터3 용으로 작성한 코드를 벡터4, 벡터11, 또는 기타 차원에도 재사용할 수 있다. 따라서 나중에 대비해 별칭을 이름으로 사용할 수도 있겠지만, 지금은 이들 차원에 1, 2, 3 등의 이름을 붙이는 데 그칠 것이다. 이는 차원에 e1, e2, e3 등의 이름을 붙이는 미분 형식 및 클리포드 대수 전통과 일치한다. e를 뺀 것만 제외하고는 말이다. 또한 우리가 벡터와 메트릭스를 참조하는 것과도 일맥상통한다(우리는 매트릭스에 정수 아래 첨자를 사용한다).

따라서 벡터3는 좌표를 알려주는 사실이 들어있는 하나의 데이터베이스가 될 것이다. 각각의 좌표는 하나의 사실로 표시되는데, 이를테면 [coordinate 1 0.337]은 "좌표 1의 값은 0.337"이라는 의미다. 그러나 다시 생각해 보면 하나의 벡터에 들어있는 모든 사실은 최초 엔트리로써의 좌표를 가지고 있을 것이므로 그 여분을 없애기만 하면 된다. 그럼 이 사실은 [1 0.337]이나 [3 -1.562]로 표시될 것이다. 실질적으로 우리가 여기서 찾는 것은 임의 데이터 요소의 튜플(tuple)이므로 지금부터 '사실'을 '튜플'이라고 부르기로 한다.

벡터3가 유효하기 위해서는 내부의 사실이 두 개의 요소 길이가 돼야 하는데, 첫 번째 요소는 정수가 되고 두 번째 요소는 부동 소수점 수가 된다(또 다른 중요한 제약이 있는데, 나중에 얘기하도록 하겠다). 오류 점검 기능 프로그램이 있어 우리가 실수할 때를 알려주면 더욱 좋을 것이다. 따라서 필자는 구조체 선언 내에서 데이터베이스 스키마를 선언하는 능력을 생성했다. 이 스키마는 데이터베이스에 저장될 수 있는 사실을 통제하고 오류 점검을 비롯해 프로그램 문서화 기능까지 제공한다. 벡터3의 경우 다음과 같이 표시된다.

```
struct Vector3  {
    [?Integer ?Float];
}
```

사실 튜플과 마찬가지로 스키마용 구문론 역시 대괄호를 사용한다. 이 스키마는 첫 번째 슬롯이 모든 정수가 될 수 있으며 두 번째는 모든 부동수가 될 수 있음을 나타낸다. 그러나 우리는 이보다 구체적이어야 한다. 벡터3의 첫 번째 좌표는 모든 정수가 될 수 없다. 1과 3 사이여야 한다. 따라서 필자는 새로운 구문론을 추가해 압축된 형식으로 이것을 선언했다.

```
struct Vector3
    [(1, 3) ?Float];
}
```

이쯤 되면 최적화된 러프 버전이 이 선언을 볼 수 있다는 사실을 쉽게 알 수 있고 벡터3 부동소수점 값을 간결하게 어레이로 저장하는 법도 알 수 있다. 항상 명심해두고 있으면 좋지만 당장은 이것에 얽매이지 않고 있다.

정수의 범위를 나타내는 ".."의 개념을 추가한 이후 명령형 표현 역시 손을 댔다. 각각의 (1..n) {...}마다 코드가 블록을 n차례 루프한다고 할 수 있다. 이것은 샘플 코드가 크레이트를 초기화하는 데 사용된다.

## 벡터3의 통사적 걸치레

지난 달 소개한 바 있는 구문론을 사용해 이 새로운 벡터3 유형을 처리할 수 있다. v라고 하는 벡터3를 가지고 있다고 가정하면 v.[1 ??]를 평가해 x 좌표를 알아낼 수 있다. 다시 말해 첫 번째 슬롯이 1인 터플의 경우 두 번째 슬롯의 값이 무엇일까? 흥미로운 (또한 C++로는 실행하기 매우 어려운) 것은 일반적인 방향에서 역행하는 조회(query)를 실행할 수 있다는 사실이다. 따라서 v.[?? 0.0]과 같은 것도 있을 수 있다. 다시 말해 좌표가 0인 목록도 있을 수 있다는 얘기다.

하지만 v.[1 ??]는 여전히 거추장스러워 보인다. C의 경우에는 v.x만 있었다. 또한 벡터의 초기화까지 염두에 둬야 한다. C의 경우 v.x = foo라고 할 수 있다. 지난 달 샘플 코드에서는 오퍼레이터 ".+"를 사용해 데이터베이스에 추가 작업을 하고 ".-"로 삭제 작업을 했다. 따라서 벡터 좌표를 1로 설정하려면 v .+ [1 foo]라고 표기하면 된다.

그러나 여기서 더욱 많은 권태를 야기할 수 있는 일관성 문제에 부딪치게 된다. v의 각 튜플이 잘 형성돼야 할 뿐 아니라 각각의 위치상 좌표마다 하나의 튜플만이 있어야 한다. v에 첫 번째 슬롯이 1인 각기 다른 개체가 있다면 문제가 된다. 따라서 위와 같이 v .+ [1 foo]를 입력하기 전에 v - [1 ?]를 입력해야 한다. 다시 말해 첫 번째 슬롯에 있는 모든 1 튜플을 삭제해야 하는 것이다. 그것을 기억하지 못할 경우 데이터의 일관성이 현격하게 저하된다. 이 요건은 귀찮을뿐더러 오류까지 유발한다. 게다가 컴파일러는 이러한 제약을 알지 못하므로 시스템이 오류를 포착할 방법도 없다.

오류 점검 기능을 향상시키고 프로그래밍 작업도 원활히 하기 위해, 필자는 도메인/범위 관계를 구조체 스키마 선언에 추가했다. 데이터베이스 접근법의 힘은 프리폼 방법으로 데이터를 조회할 수 있으므로 제약을 받지 않는다는 것이지만, 우리는 주로 (임

의 관계가 아닌) 벡터3로 기능을 모델링한다. 이 경우 한쪽 조회 방향은 전방이고 다른 방향은 후방이다. 우리는 가장 일반적인 경우인 전방 방향을 쉽게 설명하려 한다.

이 스키마에서는 심볼 "|"를 사용해 튜플의 분할을 나타낼 수 있다. 왼쪽은 모두 도메인이고 오른쪽은 모두 범위이다. 벡터3는 다음과 같이 정의된다.

```
struct Vector3
    [(1..3) | ?Float];
}
```

가끔 터플 도메인이 왼쪽에 위치하지 않을 경우도 있다. 이 경우 필자는 대체 구문을 사용해 그것을 선언한다. 관심이 있으면 샘플 코드를 참조하길 바란다.

이제 적어도 동일한 좌표에 대해 두 가지 튜플을 단언할 수 있다면 시스템은 오류 신호를 내보낼 수 있다. 그러나 필자는 C-스타일 어레이 아래첨자 구문을 사용해 명령형 표현에 일부 통사적 겉치레를 추가했다. r-값으로 v[1]을 지정하면 v.[1 ??] 표현에 해당한다. (컴파일러가 스키마를 확인하고 v 도메인이 한 가지 요소 폭이라는 것을 알기 때문이다.) v[1] = foo와 같이 v[1]을 l-값으로 사용할 경우 컴파일러는 신규 튜플 [1 foo]을 추가하기 전에 [1 ?]와 일치하는 구 튜플을 v에서 모두 삭제한다.

우수하고 간결한 구문을 사용해 더하기, 행렬 상수곱, 내적 등의 기본 벡터 처리를 정의할 수 있다. 간결하며 임의 크기의 희박 벡터(sparse vector)에서 작용하므로 상당히 깔끔하다. 희소 부분은 본 글의 범위를 넘어서는 작은 언어 추가가 필요하다. 또한 필자는 벡터3부터 벡터에 이르는 목록에서 일반화시켰다는 사실을 유념하길 바란다. 현재로써 이러한 기능의 실행에 관한 고찰은 관심 있는 독자들에게는 하나의 훈련이 될 수 있다. 필자는 만능 벡터 클래스를 이처럼 단순하고 명확하게 정의할 수 있다는 사실에 만족한다.

## 매트릭스 4

단순한 3D 애플리케이션을 기록하는 데는 벡터 뿐 아니라 매트릭스 사용까지 포함되므로, 매트릭스 정의를 살펴보기로 한다.

```
struct Matrix4
    [(1..4) (1..4) | ?Float];
}
```

두 개의 인덱스가 있으므로 튜플이 3 요소 폭이라는 점을 제외하면 벡터와 매우 유사하다. 이제, 매트릭스 m이 있는데 여기서 첫 번째 열 벡터를 끌어낸다고 가정해 보자. 데이터베이스 조회 처리는 올바른 대답만을 제공하므로 래퍼 기능 따위는 필요 없다. m.[? 1 ?]만을 평가하며 그 결과는 물음표를 채우고 있는 길이-2 터플로 이뤄진 데이터베이스다. 다시 말해 두 번째 슬롯이 1인 모든 튜플 값인 것이다. 이는 벡터4와 같은 형태지만 스타일 확인을 위해 컴파일러가 그것이 벡터4라는 것을 감지할 것인가? 그렇다는 것을 알 수 있다. 이 결과 데이터베이스에 대한 스키마를 기록한다면 그것은 두 번째 슬롯 [(1..4) | ?Float]이 없는 매트릭스4 스키마에 불과하다. 벡터4 스키마와 일치하므로, 이들 스키마를 기반으로 익명 데이터베이스의 타입-매치를 허용하면 상당히 합리적인 타입-안전을 달성할 수 있다.

또 다른 트릭도 있다. 매트릭스 m의 궤적은 + each의 m[?i ?i]일 뿐이다. 벡터와 마찬가지로 간결함을 위해 어레이 아래첨자 표기를 사용하고 있다는 사실을 주지해야 한다. 인덱스 슬롯에 동일한 가변 이름을 지정했으므로 두 슬롯의 값은 반드시 같아야 한다. 그럼 각각의 반복은 매트릭스의 대각선을 따라 이루어지게 된다. 기능 호출과 마찬가지로 러프의 수학 연산자는 자동으로 각각에 걸쳐 확장한다. 예를 들어 + each (1..10)는 55로 평가하고 * each (1..5)는 120으로 평가한다.

필자는 이 사실에 온화해진다. 이러한 표현은 매우 짧고 단순해 많은 경우 아마도 래퍼 기능을 생성하지 않아도 될 것이다. m으로부터 열 벡터를 얻으려 할 경우 get_column_vector() 기능을 호출하는 대신 m.[? n ?]만 입력하면 되는 것이다. 수학 표현을 혼합해 다른 경우에 비해 보다 유기적으로 병합할 수 있기 때문에 이것은 매우 흥미롭다.

## 어떻게 실행할 것인가?

이달의 샘플 코드를 실행해 보면 합리적인 속도로 실행된다는 것을 알 수 있을 것이다. 필자의 경우 1.5GHz 펜티엄-M 랩탑에서 초 당 30프레임을 측정한다. 언어 체계가 거의 전적으로 최적화되지 않았다는 점을 명심해야 한다. 바이트코드가 팽창돼 있고 기능 인터페이스가 느리며 무용 데이터 코드는 모든 프레임을 수집하고, 모든 데이터베이스가 여전히 연결 목록으로 실행된다. 이 모든 것에도 불구하고, 애플리케이션은 원활한 프레임 속도로 실행된다. 이것을 실행해보면 요즘 컴퓨터 속도가 엄청나게 빨라졌음을 실감할 것이다.

**목록 1. 기본 벡터 처리**

```
proc dot_product(Vector a, Vector b)
return + each a[?i] a[i] * b[i];
}
proc * (Vector v, Float factor)
Vector3 result;
each v[?i] result[i] = v[i] * factor;
return result;
}
```

글 / 스티브 데오도레

# 게임 디벨로퍼 3차 연봉 조사

올해는 진정한 게임 업계의 성숙이 이어져 오고 있지만 고통 역시 커지고 있다. 캘빈 쿨리지의 말을 빌자면 오늘날의 게임 개발 업계는 그 어느 때보다 사업의 면모를 지니고 있다. 반면 게임 개발 분야의 본질과 월가의 꾸준한 요구 사이의 격차는 점점 벌어지고 있다. 이와 함께 통합의 붐이 일면서 빈익빈 부익부 현상이 극심해지고 있다. 뿐만 아니라 미래의 콘솔과 관련한 기술적인 불확실성과 직업 안정이라는 전문적인 불확실성이 주된 화두로 부각되고 있다.

모든 환영받지 못하는 승리와 대규모 예산이 투입되는 블록버스터의 중심에는 프로그래밍, 미술 디자인, 오디오 및 프로덕션 지원 등의 비법으로 게임이라는 마법을 불러내는 개인이 있다. 올해로 세 번째가 되는 게임디벨로퍼 연봉 조사에서는 이러한 노력이 얼마나 연봉으로 이어지는지, 그리고 그것이 미국의 게임 개발자들에게 어느 정도의 생기를 불어넣고 있는지 고찰한다.

리서치 기관인 오디언스 인사이트의 도움을 얻어, 지난해 10월 게임디벨로퍼 구독자들과 2003 게임디벨로퍼 컨퍼런스 참석자, Gamasutra.com 회원 등에게 연봉 조사에 참여해 줄 것을 요청했고 전 세계에 걸쳐 4,508 건의 독자적인 응답을 얻어냈다.

모든 응답자들이 본 조사에 포함시키기에 충분한 보상 정보를 제공한 것은 아니었다. 또한 보상 금액이 만 달러 이하이거나 30만 달러 이상인 경우, 또는 보상 수치와 일치하지 않는다는 문장이 들어간 경우는 제외했다. 본 문건에서는 미국의 보상 현황만을 수록하고 있으므로 대략 1,400 건의 미국 이외 응답자들도 배제했다. 따라서 이제부터 제시하는 보상 자료에 반영된 샘플의 총 건수는 2,740 건이다.

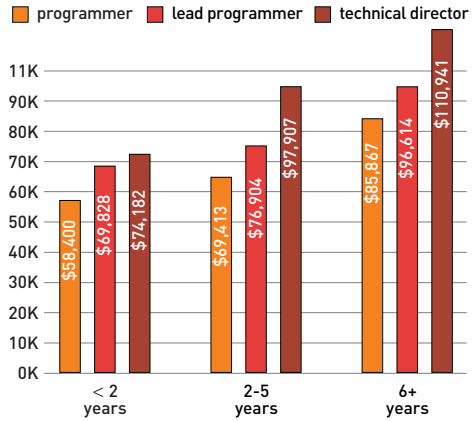
우리의 임금 조사에서 나타난 표본은 게임디벨로퍼 커뮤니티를 대상으로 했으며 신뢰도 95%에 오차 범위는 플러스 마이너스 1.8%이다. 즉 전체 인구에 걸쳐 이 정도의 오차 범위 내에서 보고된 합계 통계 중 95%가 일치한다는 의미다.

매년 게임 업계는 팬과 투기자들의 관심을 더욱 많이 받고 있다. 분석가들은 지난 몇 년 동안의 폭발적인 성장률을 예상하지 않지만, 업계 외부(특히 영화와 음악 분야)에서는 매체 연계를 통한 수익 창출 가능성을 극대화시킬 방안을 모색하고 있다. 업계 내부에서는 보다 헐리우드적인 게임 사업 모델 치환을 실험하고 있다. 여기에는 프로그래머와 아티스트, 또는 디자이너로 구성되었으며 계약이 가능한 모듈식의 훈련 중심 팀 구성이 포함된다. 게임 사업의 차후 발전 양상이 업계의 힘 균형에 영향을 미칠 것이며, 개발자에 대한 보상은 여전히 두고 봐야 할 것이다.

## 프로그래밍

게임 업계의 통합 붐과 인재 이동의 와중에서, 프로그래머들은 개발 도구, 게임 플레이, 애니메이션, 그래픽, 네트워킹, 인공 지능 또는 하드웨어 설계 등 타 개발 분야에 비해 높은 임금을 받고 있다. 그러나 차세대 콘솔이 모습을 드러내고, 스튜디오들이 원활한 이전을 도울 인재들에게 눈길을 주면서 이미 고용 시장에 미묘한 변화가 생기고 있다. 새로운 기술이 출현하면서, 기존의 인재 풀은 "발전 또는 죽음"이라는 전망에 다시 한 번 직면하게 될 것이다. 핵심적인 기술 숙련도와 더불어, 프로그래머에게 가치 있는 재산은 개발 프로젝트에서 "큰 그림"을 볼 줄 아는 유연성과, 게임 개발 업무가 프로젝트에 관한 의사 결정에 영향을 미치는 과정을 이해하는 것이다. 이러한 자질은 항상 수요가 있는 기술적 재능의 정점을 차별화 시키는 데 도움이 된다. 역경을 이겨낸 지도자나 기술 감독도 귀중하지만, 이러한 직위의 부족한 능력은 많은 일반 게임 개발자들의 발전 전망을 제한시킨다.

● 경력에 따른 프로그래밍직 급여



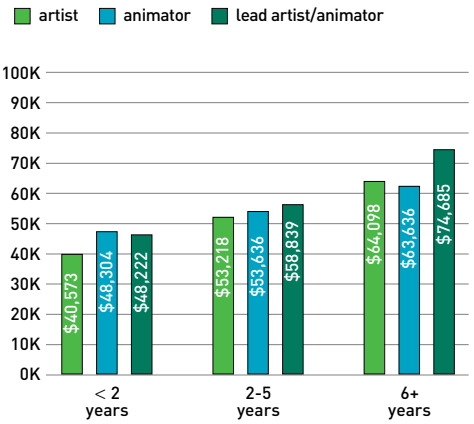

## 미술 & 애니메이션

그 어느 때보다 아트 게임에 전문성이 요구되고 있다. 종종 고용주가 충족시키기 어려운 프로그래밍 직위와 달리, 아티스트 한 명의 개방성은 수백 가지 애플리케이션을 끌어낼 수 있다. 프로그래머나 아티스트의 임금은 수요와 공급 간의 격차 정 반대를 반영하고 있는 것이다.

화가, 모델러, 애니메이터 등 아티스트 시장의 견인차는 항상 다듬어지지 않은 재능이었다. 기술 매개변수를 존중하면서도 창조적인 발전을 추진할 수 있는 아티스트나 애니메이터들이 가장 높은 보수를 받는다. 게임 업계에서 헐리우드로 유입되는 아티스트 및 애니메이터들이 늘어나면서, 이 인재들은 게임 프로젝트로 인해 자신들의 천재성에 부과되는 기술적인 한계를 반드시 따라잡아야 한다.

프로젝트 진행 과정에서 팀의 규모는 변할 수 있지만, 대부분의 게임은 여전히 선임 아티스트 한 명과 선임 애니메이터 한 명을 둔다. 콘텐츠 창조 수요가 늘어나면서 앞으로는 관리 전문 지식을 갖춘 아티스트에 대한 수요가 많아질 것이다.

● 경력에 따른 미술& 애니메이션직 급여

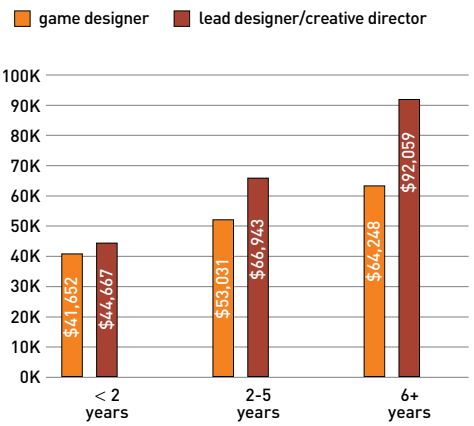


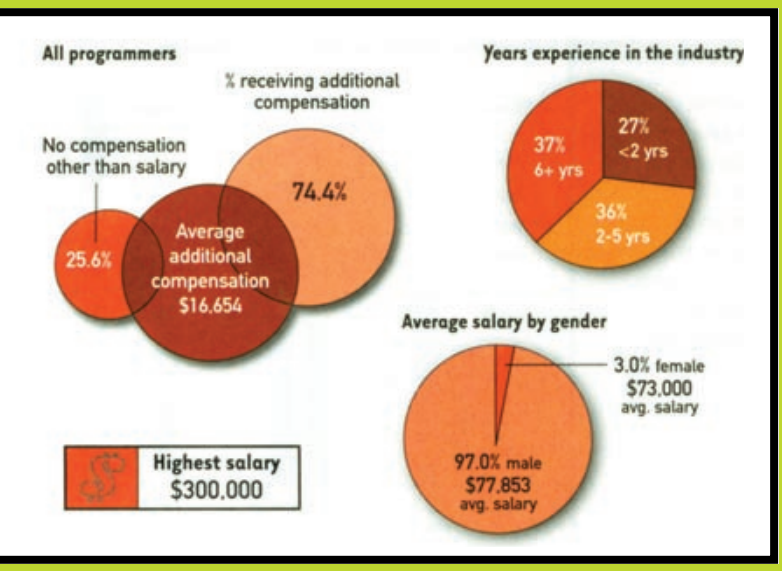

## 게임 디자인

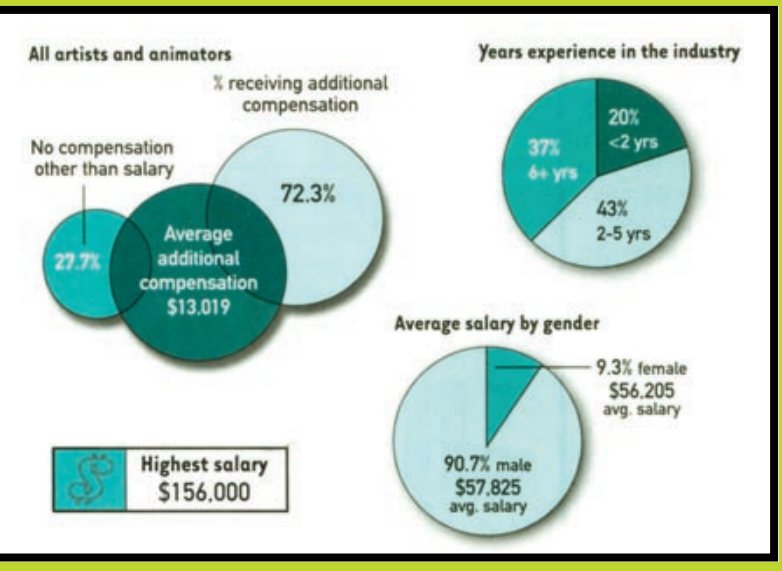
게임 디자인은 진입 경쟁이 극히 치열한 분야이며 신입 수준의 임금이 이 사실을 반영한다. 하지만 대작 타이틀을 몇 편 작업해 본 디자이너의 주가는 급속도로 치솟을 것이다. 신입 디자이너와 그보다 숙련된 디자이너, 그리고 선임 디자이너의 임금은 격차가 크다.

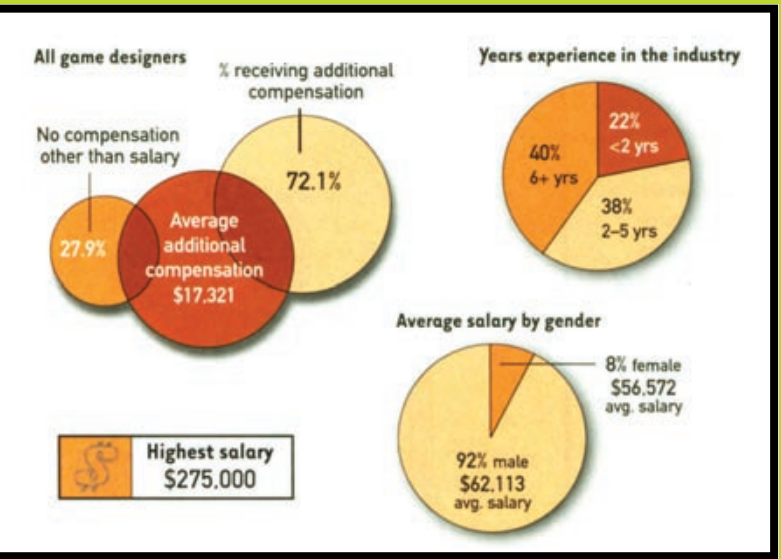
본지에서 지칭하는 "게임 디자이너"는 게임 디자이너와 레벨 디자이너, 작가들을 모두 포함한다. 게임 프로덕션 가치에 대한 소비자의 기대가 커지면서, 이제는 대본이 중요한 디자인 요소가 되었고 게임 예산 편성에 있어 보다 많은 관심을 받고 있다. 선임 디자이너 및 크리에이티브 디렉터는 대개 게임플레이 결정을 실행하는 다른 이들을 관리한다.

● 경력에 따른 게임 디자인직 급여






## 일반적인 추세

올해의 전반적인 임금 현황에는 처음으로 QA 인력이 포함되는 바람에 여러 해 전 모든 응답자들에 걸친 평균 임금과의 직접 비교가 복잡해졌다. 하지만 일부 관련 데이터는 흥미로운 경향을 보여주고 있다.

조사결과 여성 응답자들은 전체 응답자의 7%를 차지해 이전보다 소폭 증가했다. 하지만 평균 임금은 지난해 조사에서 보고된 미화 89센트보다 소폭 줄어든 87.4센트 남성 응답자들보다 지속적으로 낮았다. 그러나 미국 노동부 노동통계국은 국내에서 남녀간의 임금 격차가 지난 2002년 78센트로 소폭 줄었지만 지난 2000년의 76센트보다는 커졌다고 밝혔다.

웨스트 코스트는 계속해서 게임 개발의 온상이 되고 있으며, 고용주의 경쟁으로 인해 다른 지역에 비해 임금이 오름세를 보이고 있다. 조사에서 나타난 상위 5개 주는 캘리포니아와 워싱턴, 텍사스, 일리노이, 메사추세츠 등이다(순서 대로).

## 고용 현황: 향연과 기근

작년 연말이 가까워지면서 미국의 전반적인 고용 현황은 약간 호전된 반면, 게임 업계는 기업 통합의 파도가 몰아치면서 정리 해고와 폐업, 그리고 초보적인 차세대 전략 포지셔닝 바람이 불었다. 비벤디 유니버설 게임 R&D 부서의 마크 알자홉 채용 팀장은 게임 제작비용이 상승하자 "기업들이 실질적으로 소수의 뛰어난 인재들을 영입하려 애쓰고 있다"고 말한다. 여전히 고용주나 응시자 시장이 복잡할지 여부에 관한 문제는 양측이 서로에게 무엇을 제공해야 할지에 달려있다.

메리 마가렛 닷컴 채용 & 비즈니스 서비스 대표 겸 게임 산업 채용 담당자인 메리 마가렛 워커는 "모든 분야에 걸쳐 경쟁이 치열하며, 고객사들 역시 최상의 자격을 갖춘 취업 희망자 외에는 눈길을 주지 않는다. 취업 희망자들도 좌절하지 않고 미래의 잠재 고용주에게 더욱 많은 것을 기대한다"고 말한다.

그럼 취업 희망자는 어떤 조건을 갖춰야 최상의 자격을 얻을 수 있을까? 프로젝트에 대한 거시적인 안목을 토대로 게임 제작 업무를 이해하는 것이 상당히 유리하다. "모두가 프로그래밍에서부터 디자인에 이르기까지 제작 일정을 이해할 수 있고 공동의 목표에 정진할 수 있는 인재를 원한다"고 알자홉 팀장은 말한다. 게임 업계 채용 업체인 프리미어 서치의 질 진너 대표 말에 의하면 이제 팀웍과 유연성이 핵심 자산인데, 일부 업체들의 정리해고는 기회주의적이라고 한다. 이러한 정리 해고는 게임 업계의 경험은 많지만 신기술이나 새로운 프로덕션 모델에 적응하지 못하거나 그것을 꺼리는 사람들을 대상으로 한다. 진너 대표에 의하면 이 표류자들은

## 제작

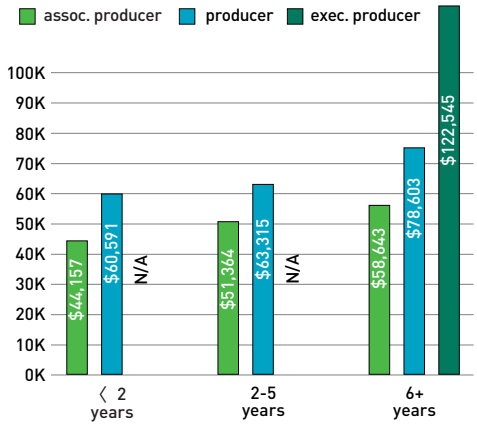
지난 몇 년 동안 게임 업계는 타 산업과 더불어 성숙기에 접어들었다. 대내외적으로 프로듀서는 필연적으로 개발 팀과 게임 제작 업무 간의 접점이다. 주로 프로젝트를 일정 및 예산에 맞춰 진행하는 의무를 맡는 프로듀서는 최종 결과물에 가장 근접해 있으므로 여타의 개발 창작 담당자들에 비해 높은 임금을 받는다.

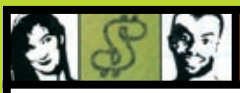
최근 몇 년 동안의 조사에 의하면 여러 타이틀의 프랜차이즈 개발 및 관리를 담당하는 숙련된 프로듀서의 평균 임금이 가장 높은 것으로 파악됐다.

스튜디오 입장에서 최고의 프로덕션 재원을 선정한다는 것은 일종의 도전이 될 수 있다.

학교에서는 제작 교육 프로그램에 초점을 두지 않으며, 이미 게임 업계에 있지 않은 이상 필요한 지식과 경험을 찾기도 힘들다. 여전히 많은 프로듀서들은 디자인 및 품질 보증 등의 단계를 거쳐 경력을 쌓아간다.

● 경력에 따른 게임 프로덕션직 급여



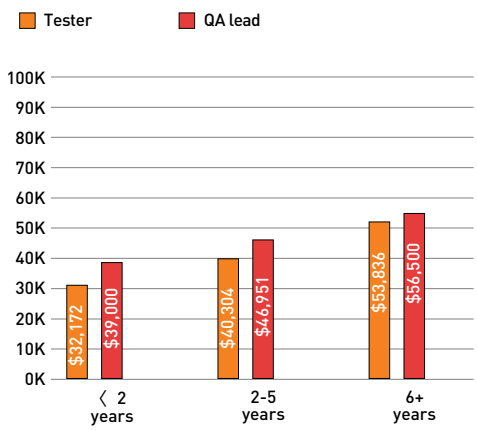
## 품질 보증

올해 처음으로 QA 부문의 임금을 포함시켰는데, 앞으로의 개발 업무를 위한 시험장으로써 테스트 부문이 담당하는 역할과, 높은 소비자의 기대를 충족시키고 반품 및 회수 비용을 최소화 시키는 데 있어 QA 기능의 중요성이 날로 커지고 있다는 점을 반영한 것이다.

테스터는 더 이상 버그 사냥이라는 제작 영역에 국한되지 않고 사용 및 포커스 그룹 테스트라는 한층 더 중요한 영역으로 확장해 나가고 있다.

14%에 약간 못 미치는 응답자들이 6년 이상 QA 부문에서 일해온 것으로 파악됐는데, 여타 부문에 비해 이 부문에서의 경험 수준이 가장 적은 부분을 차지했다. 이 수치는 한편으로 다른 개발 경력을 위한 도약대로써의 QA 역할이 과소평가된 결과이기도 하지만, 다른 한편으로는 중요성이 날로 커지고 있는 이 부문의 경험이 부족함을 의미하기도 한다.

● 경력에 따른 QA직 급여




## 오디오

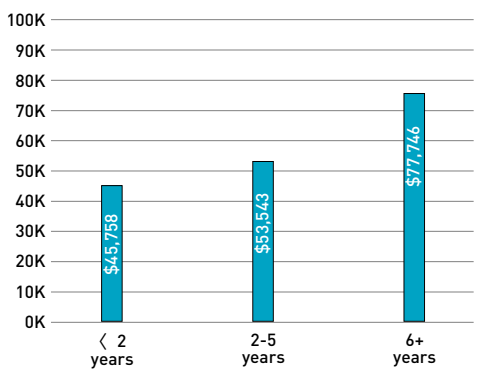
현재의 콘솔 세대는 자신들이 수년간 요구해온 것들을 오디오 커뮤니티에 제시했다.

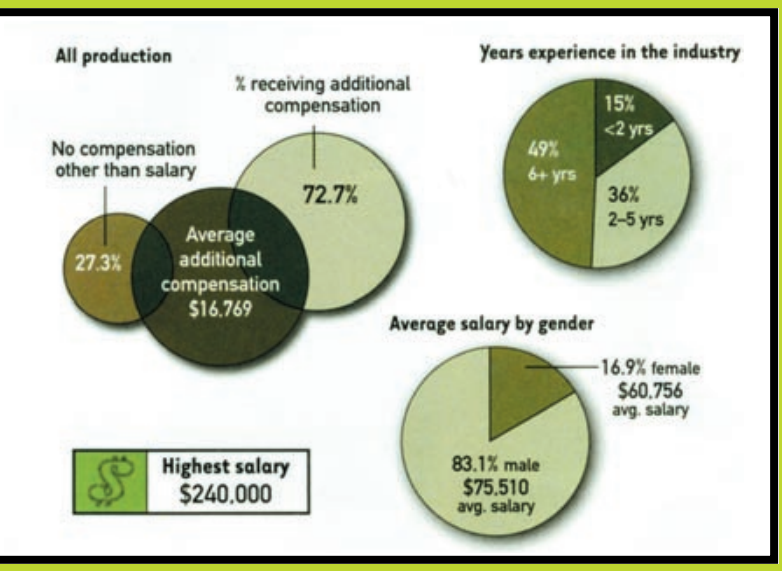
프로세서 시간과 스토리지 공간, 그리고 가장 중요한 배려가 그것이다. 홈 시어터의 인기가 치솟아 오르면서 게임 오디오 역시 데스크톱을 흔들며 짝짝거리는 PC 스피커에서 플레이어의 집 거실을 뒤흔드는 디지털 서라운드 스피커로 옮겨 가고 있다. 지금은 게임의 판촉 핵심도 돌비와 DTS 기술에 맞춰져 있으며, 올해부터는 THX까지 게임용 오디오 인증을 제공하기 시작할 것이다.

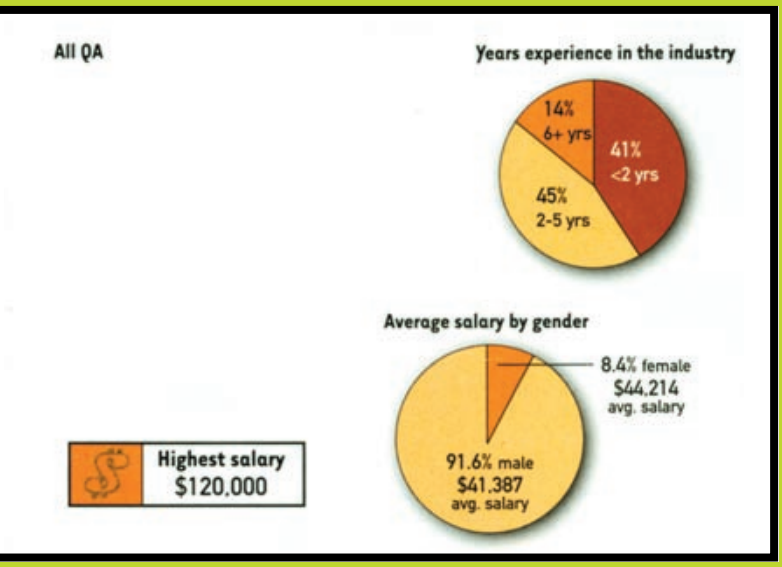
업계의 통합 덕택에 일부 대형 스튜디오에서는 규모 있는 오디오 부서를 구성할 수 있게 됐지만 상당수의 게임 오디오 인력들은 그 자리에 머무른다. 경쟁이 치열한 분야이기는 하지만 응답자의 절반이 6년 이상 일해 왔다고 답했으며, 타 개발 부서에 비해 가장 높은 편이다.

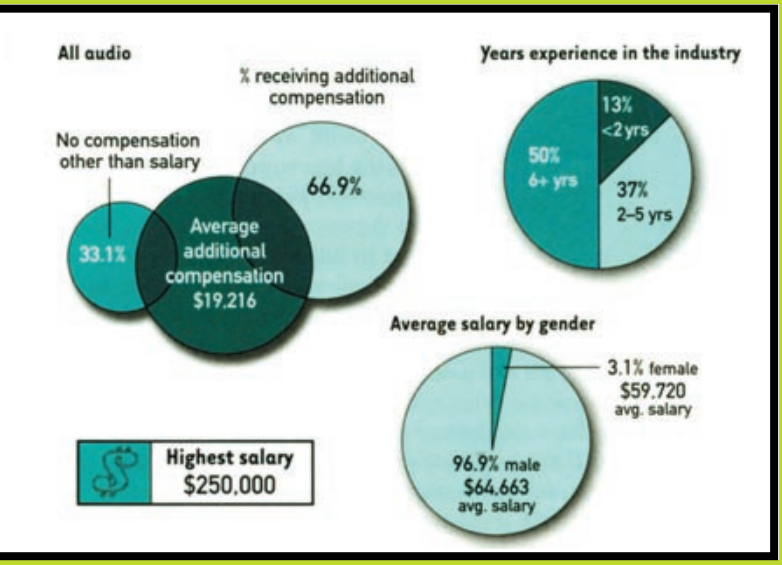
오디오 게임의 지속성에 따르는 보상이 분명히 있다.

● 경력에 따른 오디오직 급여



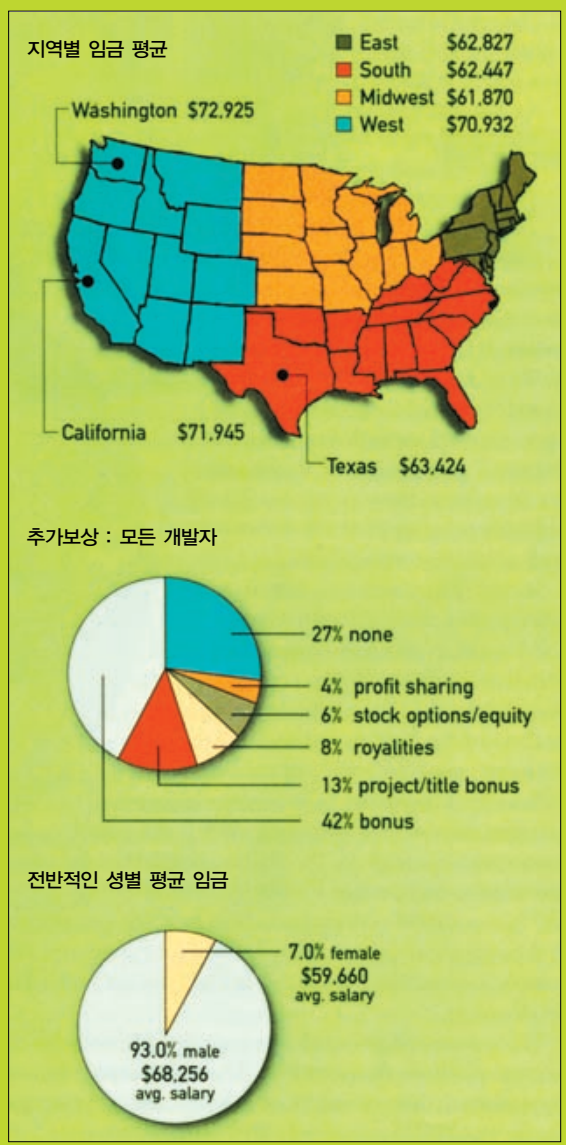



게임 사업이 성숙기에 접어들면서 점점 새 안식처를 찾기가 힘들어지고 있다고 한다. "이 사람들은 타 업종이나 업계, 또는 에듀테인먼트 업계로 유입되고는 한다. 상당수는 휴대 전화나 휴대 기기 쪽으로 옮겨갈 것"이라는 설명이다.

게다가 증가 일로에 있는 게임 연구 전문학교에서 쏟아져 나오는 그 많은 학생들이 신입 재원 시장에 어떤 영향을 미칠 것인가? 진너 대표는 "그 여파는 몇 년 후에나 느껴질 것이다. 고용주들의 경우 '게임 연구' 학교에 들어가기도 전에 학위를 가지고 있지 않으면 면담조차 하지 않으려 하는 것이 전반적인 추세"라고 말한다. 운 좋은 소수는 고용 즉시 프로그램에 투입되기도 하지만, 워커 대표는 "그러한 프로그램으로 인해 학생들이 프로그램을 성공적으로 완료한 후의 업무 선택 능력에 관해 잘못된 희망을 갖게 될까 우려된다"고 말한다.



글 / 토마스 부써

토마스 부써 | 토마스는 라이온헤드 스튜디오의 위성 개발 업체 '인터피드 게임즈'에서 개발한 타이틀 'B.C.'의 선임 프로그래머이다.

# PolySlerp

## – 빠르고 정확한 구면 선형 보간법의 다항식 근사

현재의 X박스 타이틀인 B. C.를 작업하는 동안, 우리는 이 애니메이션 시스템이 애초에 원했던 것보다 많은 CPU 시간을 소모한다는 사실을 발견했다. 헤엄치고 날고 걷는 피조물들의 수가 늘어나면서 혼합 애니메이션 수가 CPU 시간의 10% 이상을 소모한 것이다. 전산 시간을 줄이기 위해 서로 다른 아이디어를 고찰해 보았다. 멀리 그려진 캐릭터의 골격을 단순화 시키거나 각 캐릭터의 혼합 애니메이션 수를 제한하거나, 또는 하드드라이브에서 애니메이션을 초 당 30 프레임으로 스트리밍하고 근사법을 사용해 로테이션을 삽입하는 방법 등이 포함된다. 마지막 컨셉이 바로 이 글의 주제이다. 초기 요건은 가능한 한 빠르고 정확한 방법을 찾는 것이었다. 우리의 게임에는 엄청나게 큰 공룡이 등장하는데, 골격 하나의 로테이션에 작은 오류라도 생기면 바로 눈에 띈다. 또한 속도와 정확도, 그리고 메모리 풋프린트 어느 것 하나도 양보하고 싶지 않았다.

우리는 단위 4원수를 사용해 각 골격의 로테이션을 표현하므로, 다음과 같이 정의되는 고전적인 구면 선형 보간법을 사용한다.

$$\mathrm{Slerp}(q_1, q_2, t) = s(1-t)q_1 + s(t)q_2 \qquad \text{(등식 1)}$$

여기서 $s(t) = \dfrac{\sin(t\omega)}{\sin}$, $\omega = \cos^{-1} d$ and $d = q_1 \cdot q_2$

Note that $s(0) = 0$ and $s(1) = 1$

Slerp를 산출하려면 여러 가지 품이 많이 드는 삼각 함수(세 개의 사인과 한 개의 아크 코사인)를 평가해야 한다. 근사 Slerp를 시도하는 것은 바람직한 일이며 예전에도 고찰한 바 있다. 조나단 블로우는 자신의 글 "4원수 해킹"에서 quasi_slerp에 관해 주목할만한 기고를 했다(The Inner Product, 2002년 3월호). 불행하게도 2003년 3월호 게임 디벨로퍼는 사무실에서 증발해 버려서 최근에야 입수할 수 있었다. 평균적인 프로젝트의 압박을 감안해 볼 때, 이것은 배움이라는 경험을 통해 다른 식으로는 필자가 접하지 못했을 새롭고 흥미로운 기술을 습득한 행운일 것이다.

본 글에서는 우선 다항식으로 함수 s를 대체하는 방법을 논의해 본다. 새로운 보간 함수군을 PolySlerpn이라고 부르는데, 여기서 n은 s를 대체하는 다항식의 각도이다. 그런 다음 근사법에 의

한 대체로 인해 오류가 야기될 경우 그 유형과 오차 각도를 연구하고 그것을 최소화시킬 방안을 모색하도록 한다. 우연의 일치로 발견된 단순한 방법을 사용하게 되는데, 정확도를 대폭 높여준다. 또한 도표의 사용 여부, 사전 처리 단계의 필요 여부, 펜티엄 SSE 지침 설정 여부 등 각기 다른 실행 접근법을 사용할 것이다. 마지막으로 표준 Slerp 실행과 비교한 시간적 우위를 평가하고, 임의로 생성된 다량의 4원수 세트에서 유발되는 평균 오류 및 최대 오류를 수량화 할 것이다. 다음 섹션에서는 함수 s를 연구하고 우리가 추정하려는 제한 도메인을 정의하도록 한다.

## 함수 s와 근사 도메인 제한

함수 s는 필연적으로 매개변수 성격을 띠고 있으며 4원수 사이의 각도 w는 매개변수이다. 근사법을 고찰하기 전에 w 값에 따라 이 함수가 어떻게 보일지 알아야 한다. 그림 1과 2에 $\omega=\pi/2$, $\omega=3/4\pi$ 및 $\lim\omega\to0$의 경우가 나와있다.

q와 −q는 동일한 로테이션을 나타내므로 여기서 $\omega\le\pi/2$의 경우만을 고려하면 $q_1 \cdot q_2 \langle 0 (\omega \rangle \pi/2$에 해당한다)의 경우, $q_2$의 표시를 변경해 $\omega\le\pi/2$를 보장하도록 한다. 그림 1을 보면 다항식 근사법이 작업에 충분할 것이라는 점을 직관적으로 알 수 있을 것이다.

## 다항식 함수의 일반 공식

서두에서 언급한 바와 같이, 다항식 Pn을 사용하는 Slerp의 근사 방법은 다음과 같이 정의한다.

$$\mathrm{PolySlerp}_n(\mathbf{q}_1,\mathbf{q}_2,t) = P_n(1-t)\mathbf{q}_1 + P_n(t)\mathbf{q}_2 \quad \text{(등식 2)}$$

여기서 다항식 Pn의 일반 공식

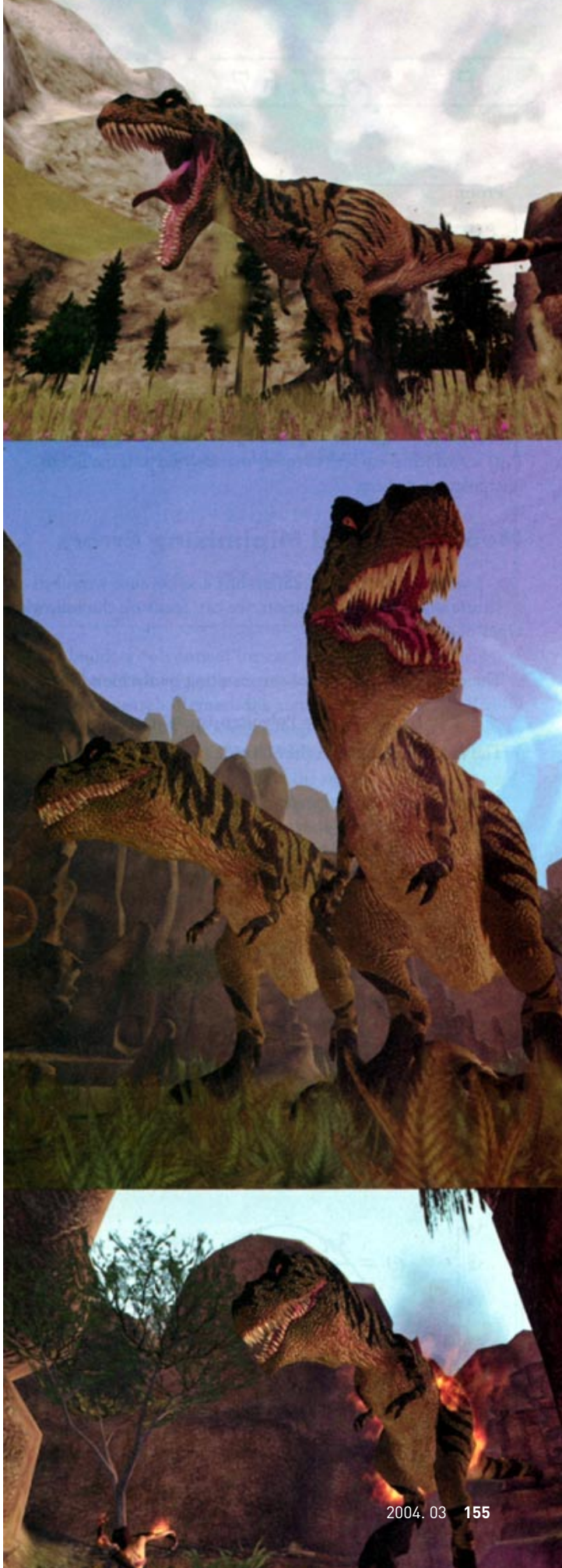
$$P_n(t) = p_{n,0} + \sum_{i=1}^{n} p_{n,i}\, t^i \quad \text{(등식 3)}$$

이제 계수 Pn,0을 Pn.n과 비교하고 다항식을 완전하게 정의하기 위해 n+1 샘플 포인트가 필요하다. 우리는 0−1을 포함하고 있는 축에 일정하게 위치한 포인트를 채택했다. s에서 채택한 샘플은 s(i/n)인데 여기서 i는 0−n을 포함하고 있는 범위이다. 우리는 Sn,i 표기를 사용해 s (i/n)을 나타낸다. 즉시 두 가지 단순화가 나타난다.

$P_n(0)= s_{n,0}= s(0)=(0)$(등식 1 참조)이므로 모든 $p_{n,0}$는 0이라고 확신할 수 있다. 즉 원점에는 상쇄가 없다.

$$P_n(0) = s_{n,0} = 0 \Leftrightarrow p_{n,0} + \sum_{i=1}^{n} p_{n,i}\, 0^i = 0 \Leftrightarrow p$$

$P_n(1)= s_{n,n}= s(1)=1$은 $P_n1$부터 $P_{n,n}$까지의 합이 1이라는 의미이며, 따라서 우리는 n−1 계수를 산출하고 다른 모든 합 중 하나에



대한 보완으로 나머지 하나를 산출할 수 있다.

증명:

$$P_n(1) = s_{n,n} = 1 \Leftrightarrow \sum^{n} p_{n,i}\,1^i = 1 \Leftrightarrow p_{n,1} = 1 - \sum^{n} p_{n,i}$$

다항식을 정리하면:

$$P_n(t) = \left(1 - \sum_{i=2}^{n} p_{n,i}\right)t + \sum_{i=2}^{n} p_{n,i}\,t^i \qquad \text{(등식 4)}$$

표 1은 세부적인 중간 분석 단계 없이 $P_2$ $P_3$ $P_4$에 대한 계수 $P_{n,2}$과 $P_{n,n}$의 공식을 수록하고 있다. 이것은 $Pn(1/n) = s_{n,1}$, $P_n(1/n) = s_{n,2}$ … $Pn((n-1)/n) = s_{n,n-1}$등식 체계를 풀어 얻어진 것이다. $P_1(t) = t$는 PolySlerp1이 선형 보간법 또는 Lerp라는 말과 동일하다는 사실에 유의해야 한다.

## 오류 측정 및 최소화

우리는 단위 4원수를 사용하고 있고 근사법으로 s를 대체하므로, 다음과 같은 오류 유형에 초점을 맞출 수 있다.

최종 4원수 길이의 오류

$$e_l = |p_s| - 1 \text{ , where } p_s = \text{PolySlerp}_n(q_1, q_2, t)$$

Slerp 각 오류

$$e_a = \cos^{-1}\left(\frac{p_s}{|p_s|} \cdot \text{Slerp}(q_1, q_2, t)\right)$$

Slerp와의 유클리드 거리

$$e_d = |p_s - \text{Slerp}(q_1, q_2, t)|$$

우리는 처음의 두 측정(길이 $e_l$과 각 오류 $e_a$의 오류) 방법을 사용해 "종합적 오류 $e_d$(유클리드 거리 오류)를 줄이는 방법을 정한다. 다른 두 개가 0이고 $e_d$ 역시 항상 0이면 종합적인 것이다.

## 길이 상의 오류 보완

길이 상의 오류 취소는 단순히 최종 4원수를 일반화시키는 문제이므로, 말하자면 4원수를 길이로 나누는 것이다. 하지만 우리의 경우에는 일반화가 약간 간소화됐다. $PolySlerp_n$의 길이는 $l = |\,a\,q_1 + b\,q_2\,|$ 이며 여기서 $a=P(1-t)$, $b=P(t)$.

우리는 $|\,q_1\,| = |\,q_2\,| = 1$이라는 사실을 알고 있는데, 이것은 길이가 다음과 같다는 의미이다.

$$l = \sqrt{a^2 + b^2 + 2abd}\,,\ d = q_1 \cdot q_2 \qquad \text{(등식 5)}$$

## 각도 오류 보완

각도 오류를 최소화하는 것은 한층 더 복잡하며, 사실 직접적인 분석 방법은 찾아내지 못했다.

간접 접근법을 고찰해 보도록 하자.

각도 오류는 Slerp에 비해 '너무 이르거나' '너무 늦은' 최종 4원수로 이해할 수 있다. 따라서 트랜스폼 t를 적용해 불일치를 보완할 수 있지만, 그 대신 계수 Pn,i를 수정해 오류를 최소화 시키는 방법을 고찰해 보자. 우리는 이 방법을 사용해 대등한 산출 비용으로 보다 정확한 결과를 얻어냈다.

## 다항식 결합을 통한 오류 최소화

분석적인 방법을 발견할 수 없었던 필자는 오류를 최소화하기 위해 (이진수나 반복 등의) 수 검색 방법을 모색했다. 예를 들어, 우리는 4원수 간의 각도에 따라 $P_{n,i}$의 값으로써 다항식으로 $P_{n,i}$ 계수를 근사할 수 있었지만, 분명 검색을 위한 자유 변수의 수는 n이 증가하거나 다항식 근사 $P_{n,i}$의 각 k가 높아짐에 따라 급속도로 엄청나졌다. 각도 k의 다항식으로 $P_{n,2}$ 모두를 근사할 경우, 검색을 위해 (n−1)(k+1) 변수가 필요하다. 따라서 우리는 다른

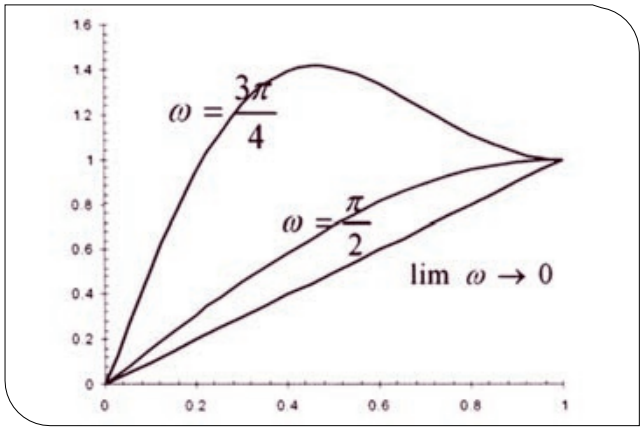



그림 1. 각기 다른 w 값에 대한 S(t)

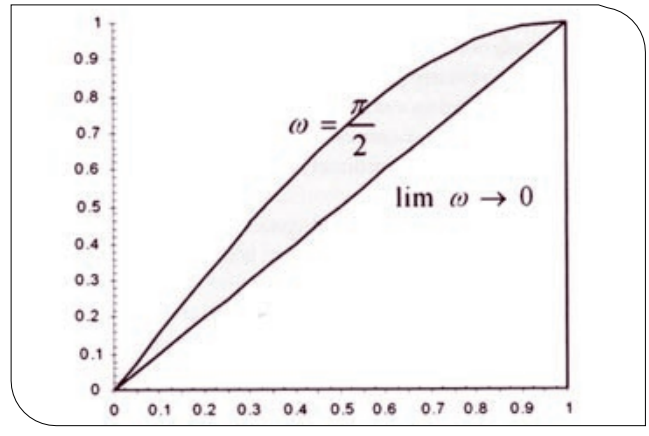



그림 2. 대상 도메인

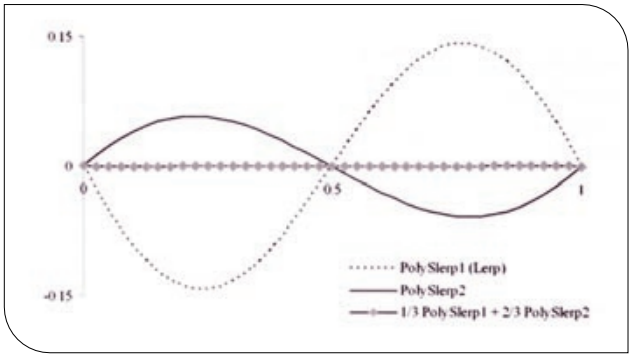



그림 3. w=π/2의 경우 함수 t의 각도 오류

접근법을 시도했다.

이 방법의 원칙을 설명하기 위해, 우선 간단한 예를 들어보자. 그림 3에 $\omega = \pi/2$의 경우 선형 보간법($PolySlerp_1$) 및 $PolySlerp_2$에 의해 t에 걸쳐 생성된 오류가 나와있다. $PolySlerp_2$에 의한 오류가 대략 $PolySlerp_1$에 의한 오류의 절반이 된다는 사실을 직감할 수 있다. 각도 오류를 최소화하기 위해 선형 구조인 $1/3P_1+2/3P_2$로 $P_2$를 교체하기에 충분한지 의아할 것이다. 그림 2의 세 번째 곡선을 보면 알겠지만 얼핏 보면 흥미로운 옵션처럼 보인다. $\omega = \pi/2$와 같은 특정 경우에는 직관적인 비율일 뿐이지만 이것을 통해 우리의 접근법을 제시할 수 있다. 1도 다항식의 선형 구조를 사용해 오류를 최소화시키는 것이다. 상수 계수가 모든 $\omega$ 값에 충분한 것으로 밝혀졌다. 최종 구조는 $P_n$과 같은 각도가 되지만 계수는 다른 것이다. 우리는 이 새로운 다항식을 다음과 같이 정의한다.

$$Q_n(t) = \sum_{i=1}^{n} m_{n,i} P_i(t) \qquad \text{(등식 6)}$$

여기서 $m_{n,i}$는 $P_i$에 적용된 계수이다.

표 1. pn,2–pn,n 공식화

| P | Coefficients |
|---|---|
| $P_2$ | $p_{2,2} = 2 - 4s_{2,1}$ or simplified as $p_{2,2} = 2 - \frac{2\sqrt{2}}{\sqrt{d+1}}$ |
| $P_3$ | $p_{3,2} = -\frac{9}{2}(5s_{3,1} - 4s_{3,2} + 1)$ |
| | $p_{3,3} = \frac{9}{2}(3s_{3,1} - 3s_{3,2} + 1)$ |
| $P_4$ | $p_{4,2} = -\frac{2}{3}(104s_{4,1} - 114s_{4,2} + 56s_{4,3} - 11)$ |
| | $p_{4,3} = \frac{16}{3}(18s_{4,1} - 24s_{4,2} + 14s_{4,3} - 3)$ |
| | $p_{4,3} = -\frac{32}{3}(4s_{4,1} - 6s_{4,2} + 4s_{4,3} - 1)$ |

이 방법은 결과가 일반화 될 것이라는 가정 하에서만 유효하다. 단순하게 $1/3P_1+2/3P_2$의 경우 $PolySlerp_1$이 4원수의 길이를 대폭 변경했음을 즉시 알 수 있다. $m_{n,1}$의 합이 모든 임의 값(0 제외)이 될 수 있음을 암시하는 보간법 말미에 다항식을 일반화시키고 이 합이 항상 1이 되도록 제약을 둔다. 이렇게 하면 검색을 위한 $m_{n,i}$의 수가 하나로 줄어들어 다른 모든 $m_{n,i}$의 합계 중 하나를 간단히 보완할 수 있게 된다. 앞서 제산한 방법에 비해 검색을 위해서는 n−1 매개변수만이 필요할 뿐이다. $q_{n,i}$ 중 i 합은 1이라는 사실에 주목할 필요가 있다.

모든 선형 구조를 간소화시킨 후, 우리는 표 2에 나와 있는 새로운 $q_{n,i}$ 계수를 구했다. 이제는 오류를 최소화시켜 $m_{n,i}$ 계수를 근사하기 위해 '언덕 오르기' 유형의 함수만이 필요할 뿐이다. 최대 각도 오류($e_{a\ max}$)뿐 아니라 평균 각도 오류($e_{a\ avg}$)까지 최소

표 2. 간결화된 Qn,i 계수

| Q | Coefficients |
|---|---|
| $Q_2$ | $q_{2,2} = m_{2,2}(2 - 4s_{2,1})$ |
| $Q_3$ | $q_{3,2} = m_{3,2}(2 - 4s_{2,1}) - \frac{9}{2}m_{3,3}(5s_{3,1} - 4s_{3,2} + 1)$ |
| | $q_{3,3} = \frac{9}{2}m_{3,3}(3s_{3,1} - 3s_{3,2} + 1)$ |
| $Q_4$ | $q_{4,2} = m_{4,2}(2 - 4s_{2,1}) - \frac{9}{2}m_{4,3}(5s_{3,1} - 4s_{3,2} + 1) - \frac{2}{3}m_{4,4}(104s_{4,1} - 114s_{4,2} + 56s_{4,3} - 11)$ |
| | $q_{4,3} = \frac{9}{2}m_{4,3}(3s_{3,1} - 3s_{3,2} + 1) + \frac{16}{3}m_{4,4}(18s_{4,1} - 24s_{4,2} + 14s_{4,3} - 3)$ |
| | $q_{4,4} = -\frac{32}{3}m_{4,4}(4s_{4,1} - 6s_{4,2} + 4s_{4,3} - 1)$ |

화시키려 한다. 이를 위해 실제로 $e = e_{a\ max} + k_a e_{a\ avg}$를 최소화시키고 평균 및 실제 각도 오류 사이에서 관찰된 실제 통계 비율을 감안하기 위해 검색 과정에서 $k_a$를 갱신한다.

## 실행

여기서는 각기 다른 접근법을 고찰해 보도록 한다. $PolySlerp_2$와 오류 보완 버전의 경우, ($p_2$를 정의하기 위한) $P_{2,2}$ 및 ($Q_2$를 정의하기 위한) $q_{2,2}$ 계수는 실시간이나 룩업 테이블에 저장된 내적 d로 직업 평가할 수 있다.

$$p_{2,2} = 2 - \frac{2\sqrt{2}}{\sqrt{d+1}}$$

$$q_{2,2} = m_{2,2}\left(2 - \frac{2\sqrt{2}}{\sqrt{d+1}}\right)$$

주: m2,2는 검색 알고리즘을 통해 발견된 상수이다.

또 다른 옵션은 4원수 쌍과 함께 내적 d와 계수 $P_{2,2}$ (또는 $q_{2,2}$)를 저장하는 것이다. 마지막으로, 계수 $P_{2,2}$ (또는 $q_{2,2}$)를 담고 있는 룩업 테이블을 사용할 수 있다. 이 테이블의 인덱스는 테이블의 크기와 곱한 내적이며, 우리는 이 인덱스를 "다항식 ID" 또는 간단히 줄여 PolyID라고 부른다. B.C.의 경우에는 2,048개의 엔트리가 들어있는 테이블을 사용한다.

하지만 $PolySlerp_3$와 $PolySlerp_4$ 및 오류 보완 버전의 경우 계수 $P_3$, $Q_3$, $P_4$ 및 $Q_4$는 s에 관해 여러 개의 샘플 포인트를 필요로 하며, 바로 이러한 이유로 룩업 테이블에 저장해야 한다.

$q_1$과 $q_2$가 미리 알려진 경우(대개 애니메이션이 여기에 해당된다) 4원수 쌍에 통상 익스포터 코드로 PolyID를 추가한다. 재 일반화를 위해서는 내적이 필요하지만(등식 5 참조) 테이블에 저장할 수 있으므로 실시간으로 계산할 필요는 없다. 이 경우 일반화가 완전히 정확하지는 않지만 합리적인 trade-off로 간주한다.

(예를 들면 두 가지 골격 자세를 혼합하는 동안) $q_1$과 $q_2$가 미리 알려지지 않은 경우 PolyID를 계산해야 하며, 또는 앞서 설명한 바와 같이 $PolySlerp_2$의 경우에는 삽입 코드로 직접 다항식의 계수를 산출할 수 있다.

$PolySlerp_2$의 경우 코드는 직접 버전과 룩업 테이블 사용 버전 모두 제공된다. 우리는 오류 수준과 두 버전의 속도를 비교하기 위해 이것들을 코딩한다.

우리는 모든 $PolySlerp_n$을 표준 버전과 SSE 버전으로 실행했다. SSE 코드는 "역 제곱근 근사"(기억하기 좋게 "rsqrss"라고도 한다)를 사용한다는 점을 주지해야 하는데, 훨씬 빠르기는 하지만 $1/\sqrt{x}$를 산출하는 표준 방법에 비해 정확도가 떨어진다. 이것은 새로운 보간법 불일치의 원인이 된다. 우리는 또 다른 $m_{n,i}$ 계수 세트로 이것을 상당 부분 보완했다. 따라서 $m_{n,i}$를 산출하는 오류 최소화 함수는 표준 또는 SSE 모드로 컴파일해야 한다.

## 통계: 오류 및 속도

마지막으로 이 방법이 그만큼의 가치가 있는지 살펴보기로 하겠

표3. 표준 버전의 보간법 오류 테스트 결과

| METHOD | VERSION | TIME | ANGULAR | LENGTH X1000 |
|---|---|---|---|---|
| D3DSlerp | From the standard Xbox library | 221 | 0.00 | 0.00 |
| Lerp | | 60 | 1.100 | 104.85 |
| Lerp | Err. Compensated | 70 | " | 0.09 |
| PolySlerp4 | Standard | 85 | 0.006 | 0.04 |
| | Preprocessed PolyId | 67 | " | " |
| | Preprocessed PolyId + Err. Compensated | 83 | " | 0.09 |
| | Err. Compensated | 106 | " | " |
| PolySlerp3 | Standard | 84 | 0.013 | 0.58 |
| | Preprocessed PolyId | 68 | " | " |
| | Preprocessed PolyId + Err. Compensated | 74 | 0.006 | 0.09 |
| | Err. Compensated | 93 | " | " |
| PolySlerp2 | Preprocessed PolyId | 55 | 0.471 2.09 | |
| | Preprocessed PolyId + Err. Compensated | 71 | 0.008 | 0.09 |
| | Normalized | 88 | " | " |
| | Err. Compensated + No lookup table | 74 | " | " |
| | Err. Compensated + Coefficients stored in animation key | 63 | " | " |

다. 우리는 20만 쌍의 4원수 삽입 시간을 측정하고 각도 및 길이의 평균 오류를 Slerp와 비교해 평가했다. SSE 확장을 활성화시킨 상태에서 표준 버전으로 X박스를 테스트했다. 표준 버전의 경우 표3과 같은 그림을 얻을 수 있었다.

$PolySlerp_4$는 오류 보완이 없이 상당히 정확하므로 애니메이션 키 프레임(사전 처리된 POlyID)에 삽입하거나 일반 목적으로 Slerp(표준 버전)를 대체하기 좋을 것으로 보인다. 이제는 룩업 테이블 사용이 문제가 될 수 있다. 매우 비우호적으로 저장될 수 있고 두 룩업 테이블 간의 단절을 유발할 수도 있다. 이 경우에는 "Err. Compensated + No look-up table" 버전이 사용될 수 있다.

표 4에 SSE 실행 결과가 나와있다. PolySlerp의 속도가 향상됐음을 알 수 있고 "역 제곱근 근사" 어셈블러 오퍼레이터가 정확도에 미치는 영향도 측정할 수 있다. $PolySlerp_4$의 오류 보완이 실제로는 SSE에 이로울 것이 없다는 것이 분명해졌다. 길이에 있어서의 평균 오류를 배가시킬 뿐이다. 이 표를 보면 애니메이션 키 프레임 보간법에는 $PolySlerp_4$ 중 "사전 처리 PolyID" 버전이 적합하고 다른 경우에는 $PolySlerp_2$ 중 "Err. Compensated + No lookup table" 버전이 적합한 것으로 보인다. 애니메이션 데이터에 사용된 더 많은 메모리 비용으로 오류 보완 + 애니메이션 키에 저장된 계수를 사용할 수 있다.

## 정확도 trade-off

애니메이션 기법을 향상시켜 콘텐츠를 보다 풍부하게 만드는 일은 이 업계의 지속적인 관심사이다. 보다 생동감 있는 동작을 위해 더 많은 애니메이션을 혼합하건 보다 복잡한 캐릭터 골격을 구축해 현실감을 높이건, 또는 화면에 보다 많은 배우를 등장시켜 보다 흥미로운 세계를 창조하건 간에, 전산 비용이 급격하게 늘어나고 있다. 메모리와 정확도의 trade-off가 원활하다면 빠른 애니메이션 삽입용 근사법은 유용한 도구라는 것이 입증되었다. 많은 경우 일부 수정 과정이 들어간 Lerp 함수도 충분히 정확하지만 오류가 눈에 띌 정도로 축적되면 보다 정확한 방법이 필요하다. 바로 이 때문에 PolySlerp가 우리에게는 매우 유용했고 다른 개발자들에게도 유용할 것이다.


## 감사의 말

수 년 동안 시스템 분석과 알고리즘 설계에 기여해 온 말리카 살루니어와 이 글을 숙독하고 건설적인 조언을 아끼지 않은 앤드류 비들러, 샘 마틴에게 감사의 말을 전한다.


표4. SSE를 활용할 경우의 보간법 오류 테스트 결과

| METHOD | VERSION | TIME | ANGULAR | LENGTH X1000 |
|---|---|---|---|---|
| D3DSlerp | From the standard Xbox library | 221 | 0.00 | 0.00 |
| Lerp | | 60 | 1.100 | 104.85 |
| Lerp | Err. Compensated | 70 | " | 0.09 |
| PolySlerp4 | Standard | 85 | 0.006 | 0.04 |
| | Preprocessed PolyId | 67 | " | " |
| | Preprocessed PolyId + Err. Compensated | 83 | " | 0.09 |
| | Err. Compensated | 106 | " | " |
| PolySlerp3 | Standard | 84 | 0.013 | 0.58 |
| | Preprocessed PolyId | 68 | " | " |
| | Preprocessed PolyId + Err. Compensated | 74 | 0.006 | 0.09 |
| | Err. Compensated | 93 | " | " |
| PolySlerp2 | Preprocessed PolyId | 55 | 0.471 | 2.09 |
| | Preprocessed PolyId + Err. Compensated | 71 | 0.008 | 0.09 |
| | Normalized | 88 | " | " |
| | Err. Compensated + No lookup table | 74 | " | " |
| | Err. Compensated + Coefficients stored in animation key | 63 | " | " |

글 / 모건 W. 게리

# 토털리게임즈의 '노르망디 상공의 비밀 병기'

토털리게임즈는 사상 최대의 충돌 사건인 2차 세계 대전과 자랑스러운 연관을 맺고 있다. 대표작인 '배틀 호크'와 호평을 얻고 있는 '독일 공군의 비밀 병기'는 이전의 습격을 주제로 삼았다. 또한 엑스 윙과 TIE 파이터를 선보인 후 스타 트랙: 비릿지 커멘더에 이르기까지 은하계처럼 멀고 먼 여정을 거쳐온 것으로도 유명하다. 그러나 자세히 들여다보면 실제의 역사적 사건을 다루는 데 있어 더욱 큰 만족감을 일으키는 절대적인 뭔가가 있다. 노르망디 상공의 비밀 병기(SWON)는 토털리게임즈가 처음 시작했으며, 루카스아트는 2차 세계 대전을 무대로 설정하기를 바랬다. 라이언 일병 구하기 등의 영화와 (스테판 앰브로스의 훌륭한 저서를 원작으로 한) 밴드 오브 브라더스 등의 텔레비전 쇼, 그리고 영광의 훈장 시리즈 같은 비디오 게임들은 전쟁이 끝난 지 50년도 넘었지만 여전히 많은 이들의 마음을 울리고 있음을 의심하지 않게 해준다. 토털리게임즈 역시 2차 세계대전 시기의 공중전 스타일을 재현할 적기라고 판단했다. 또한 우리의 게임 스타일을 콘솔 이용자들에게 소개한다는 사실에 흥분되기도 했는데, 업체에나 콘솔 시장 모두 처음 있는 일이었다. 디테일이 정해지자 곧장 본격적인 개발에 착수했다. 본 글을 통해 교차 플랫폼 개발과 관련한 몇 가지 문제들을 설명하고자 한다. 플레이스테이션 2와 X박스, PC 버전의 동시 발매는 SWON이 선을 보인 후 가장 독특하고 새로운 도전이었다. 발매 일자가 정해지자 우리는 지금까지 작업해야 했던 범위 내에서 신속하게 박스를 지정했다. 우리는 전형적인 게임 개발자들로써 said box를 채우기 위해 모든 노력을 기울였고, 가득차면 그것을 새로운 추가 사항에 맞게 늘려나갔다. 그런 후 가용한 시간에 맞춰 게임 플레이, 기술 특성 및 기타 잡다한 것들과 관련한 모든 결정들을 평가했다. 범위에 너무 주의를 기울일 경우 항상 위험이 도사리게 마련이다. 지나치게 많이 나가는 바람에 자료가 누락될 경우 진열대에서 완제품을 볼 수 없게 될 위험이 있다. 너무 멀리만 나가지 않으면 평범한 타이틀은 낼 수 있다. 처음서부터 막판까지 우리는 이 문제에 초점을 맞췄다. 결국은 일종의 도박으로 실현됐다.

## 올바른 선택

**1. 미들웨어.** 우리는 처음부터 일정을 감안해서 완전히 새로운 엔진을 만드는 것은 옵션이 아니라는 것을 알았다. 엔지니어들이 엔진 작업을 하는 동안 적극적인 개발을 중단할 형편이 못됐다. 따라서 미들웨어에 구조를 요청할 수밖에 없었다. 그 당시의 교차 플랫폼 미들웨어 솔루션을 면밀히 살펴본 후 '크리테리언스 렌더웨어'를 선택하기로 결정했다. 토털리 게임의 역사서에도 이 결정이 탁월했던 것으로 적혀있다.

렌더링 업체와 여러 가지 지원 시스템 외에도, 미들웨어 솔루션의 선택은 자산 체인 및 파이프라인의 토대를 제공해 주기도 했다. 이는 상용 기성품 엔진을 구입할 때 종종 간과되는 측면이다. 초기에 이처럼 유리한 고지를 선점한 덕분에 대다수 프로젝트 특유의 초기 난관을 피할 수 있었다. 기록적인 시간 내에 기초 물리학과 인공 지능을 겸비한 그래픽을 화면에 띄웠을 뿐 아니라 모든 플랫폼에 걸쳐 전체 패키지를 CD로 부팅할 수 있었다.


게 임 데 이 터

**배급:** 루카스아트

**상근 개발자 수:** 상근 개발자 24명, 수시 파트 타임 개발자 4~5명, 루카스 아트의 사운드, 음성, 현지화 및 QA 부서

**계약직:** 미술, 각본, 성우

**개발 기간:** 18개월

**발매일:** 2003년 11월 18일

**대상 플랫폼:** 플레이스테이션 2, X박스, PC

**개발 하드웨어:** 256-1024-MB RAM을 탑재한 펜티엄 1-2.4GHz 기기와 다양한 그래픽 카드

**개발 소프트웨어:** 렌더웨어, CRI, 3DS 맥스, 포토샵, 코드 워리어, MS 비주얼 스튜디오, 다양한 내부 전용 툴

**프로젝트 규모:** 파일: 30,284; 코드: 246,513 + ~85K 실제 코드 라인

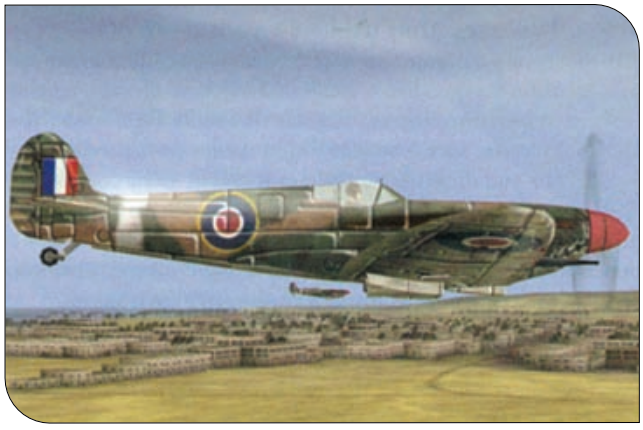
물론 게임을 여러분의 것으로 만들어줄 미들웨어 솔루션은 없다. SWON의 비전을 실현시키기 위해 효과 시스템과 지형 렌더링, 물리학을 비롯한 기타 수많은 렌더웨어의 관점에서 상당한 작업이 수행됐다. 그러나 결국 우리가 개발 주기를 마칠 때까지 자신감을 갖고 프로젝트를 진행할 수 있도록 한 것은 렌더웨어가 제공한 초기의 토대였다. 게임의 업 앤 러닝(up and running)이 빠를수록 디자인 이론을 증명하거나 수정이 필요한 러프 스폿을 파악할 수 있다. 미들웨어를 채택한 덕분에 이러한 문제들이 불거져 나왔을 때 그것을 처리할 시간을 벌 수 있었다.

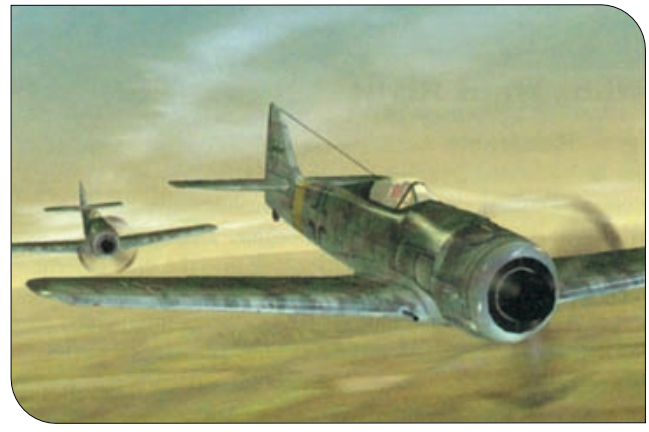
**2. 팀.** 진정으로 가장 중요한 게임 개발 부분이 바로 팀이었다. 전체 개발 과정에서 우리 팀은 마치 윤활유를 친 기계처럼 원활하게 작업했다. 토털 게임즈의 규모 역시 일종의 재산이었다. 팀이 작으므로 가끔 게임 업체에서 볼 수 있는 전통적인 구분이 없었다. 프로그래머는 디자이너와, 아티스트는 프로그래머와 격의 없이 의사소통했다. 일반적으로 팀 내의 모든 구성원이 사이좋게 지내는 상황이 만들어지기를 기대하기는 힘들다. 하지만 문제나 아이디어가 팀 전체를 통해 순환 및 존중되는 협력적인 여건을 조성할 수는 있다. 우리는 내부 팀과 더불어 외부 파트너와도 유사한 관계를 조성했다. 사운드 디자인과 음성 지원, 추가 미술 리소스, 현지화 지원 및 품질 인증의 경우 루카스 아트에 의존했다. 결국에는 응집력 있는 게임을 만드는 데 도움이 되는 종합체의 일원으로 모든 것을 취급했다. 사실 여러 차례의 열띤 논의가 있었지만 모두 더 나은 게임을 만들기 위한 열정에서 비롯된 것들이다. 비생산적인 논쟁이나 개성 충돌은 프로젝트 시작부터 애당초 존재하지 않았다. 전체적으로 게임을 만들기에 더 없이 좋은 여건이었다.

**3. 디자인 초점.** 프로젝트 초기부터 팀 전체가 이제 만들려는 게임에 관한 분명한 아이디어를 가지고 있었다. 바로 역사에서 영감을 얻어 2차 세계 대전 공중적 액션 타이틀을 만드는 것이었다. 이것은 목표이기도 했다. 모든 이들을 창조적으로 한 자리에 모이게 하는 일은 간과할 수 없다. 기술이나 공정 및 전반적인 개발에 이르자 난관에 봉착하기도 했지만, 팀원 모두가 최종으로 게임이 갖추게 될 외양과 플레이 방식을 확실히 이해하고 있었다. 이것을 염두에 둔 덕분에 그 목표에서 벗어나는 게임 플레이를 고집하거나 위험천만한 실험을 하는 등의 불상사는 없었다. 잘 정의된 범위 내에서, 우리는 전투 액션의 플레이 방법을 정의 및 향상시키는 쪽으로 모두의 창조적인 에너지를 모을 수 있었다.

**4. 활발한 프리프로덕션.** 토털리게임즈는 처음으로 콘솔 시장에 진출하는 입장이었고 세 가지 플랫폼에 걸친 동시 발매도 처음 있는 일이었으므로, 모든 것을 한 데 결집시킬 수 있는 길은 견고한 기획과 현실적인 일정 뿐이라는 사실을 우리 모두 알고 있었다. 스케줄링은 대부분 마이크로소프트 프로젝트로 진행됐다. 시작 및 마감 일자를 비롯해 여러 가지 지침을 제시하고 신속하게 작업해야 할 시간대까지 정했다. (기술, 미술, 디자인) 팀장들에게 자체 일정을 마련하라고 지시한 후 나중에 그것

모건 W. 그레이 | 모건은 5년 가까이 토털리 게임즈에서 일했다. SWON에서는 코디네이터 역할을 맡았다. 그 이전에는 엑스 윙 대 TIE 파이터: 밸런스 오브 파워, 엑스 윙 연합군, 스타 트랙: 브릿지 커맨더의 미션 빌더/레벨 디자이너를 맡았고, 시네마웨어/캡콤에서 만든 로빈훗: 왕의 수호자에서 선임 게임 디자이너를 맡기도 했다.





을 조합해 마스터 스케줄을 짰다. 팀장들에게 일방적인 표준 형식을 사용하도록 강요하기보다는 자신에게 편한 방법을 사용할 수 있도록 허용하는 것이 훨씬 효율적이었다. 게임 개발은 구조적 특성상 가장 잘 알려진 스케줄링 유형에 저항하는 것으로 비쳐지는 독특한 창작 작업이다. 그럼에도 불구하고, 비록 높은 수준의 커뮤니케이션이 요구되긴 했지만 우리의 방법은 탁월한 효과를 거두었다. "원대한 비전"을 갖고 있을 뿐 아니라 마감 시한까지도 신경 쓰는 사람을 통해 모든 조각들을 한 데 맞출 수 있었다.

우리는 또한 프리프로덕션 기간을 활용해 세부적인 수단들을 처리했다. 여러 가지 테스트를 생성해 시스템을 강조하기 시작했다. 개발 과정에서 실제 테스트에 응용한 결과 전반적인 생산성을 크게 증대시킨 명백한 사안들을 모색할 시간을 확보할 수 있었다(나중에 설명할 미술 팀은 제외). 또한 이 시간을 활용해 여러 가지 기술 조각들을 테스트했는데, 그 중에서도 지형 시스템이 두드러졌다. 마지막으로 핵심 디자인을 문서화했는데, 게임의 흐름을 좌우하는 견고한 로드맵을 제공했다. 더불어, 성숙한 디자인을 보유한 덕분에 자산 목록이라든가 인공 지능 요건을 비롯한 기타 잡다한 게임 요소들을 생성해 전반적인 스케줄을 추가로 손질할 수 있었다.

**5. build 공정.** 대부분의 게임 개발자들과 마찬가지로, 우리 역시 생산적인 개발의 토대는 도구 및 소통망을 최종적으로 결정하는 것이라고 믿었다. 모델 익스포터나 미션 스크립팅 유틸리티만큼 화려하지는 않지만, 우리의 구축 공정은 우리의 모든 노력이 부팅 가능 버전이라는 영광과 위엄을 통해 실현될 수 있게 해준 견인차였다. 여러 가지 구성 및 출력 스타일을 사용자가 선택할 수 있다는 점에서 우리의 공정은 "자동화" 됐으며, 이른바 "빅 버튼"으로 알려진 것을 타파하는 것으로 공정을 시작했다. 불행하게도 우리는 프로젝트 과정에서 팀의 누군가가 make build를 처리할 수 있는 곳까지 닿을 수 없었기 때문에 "자동화"라는 단어에 인용 부호를 쓸 수밖에 없었다. 이것은 원활하게 작동하던 툴 자체로 인한 것이 아니라 프로젝트가 끝나갈 때까지 우리에게 "normal build"가 없었기 때문이다. 불가피했던 것은 수정을 위해서는 전문 build 지식이 필요한 코드나 asset bank가 깨지는 것이었다. 메인 build keeper가 이 영역에 열쇠를 보관함에 따라, 우리 중 누구도 매일 매일의 사안 대부분에 손을 댈 수 없었다. 대다수 게임 제작 측면과 마찬가지로 이러한 전문 지식은 팀 전반에 보급돼야 한다. 팀원 중 한 명만이 특수한 작업을 처리할 수 있다면 그 사람이 병에 걸리거나 휴가를 떠나거나 교통사고를 당할 경우 시간을 낭비하게 될 공산이 크다.

프로젝트 초반에 build가 닫히는 바람에 복구하기까지 7시간이 걸린 적이 있다. 이 때문에 업데이트와 수리가 모두 완료될 때까지 그 날 하루 종일 업무가 지체됐다. 여러 팀원들의 노력 덕분에 전반적인 build 시간을 약 두 시간 반으로 줄일 수 있었다. 이렇게 하면 게임을 신속하게 만들어 개발 팀원들과 품질 인증 팀원들에게 배포할 수 있다. 여러 가지 build 전용 기기를 사용해 이제는 세 가지 플랫폼에 걸쳐 현지화 버전과 함께 게임을 돌려보는 데 6시간이 채 걸리지 않게 됐다. 이러한 속도 덕분에 기록적인 시간 내에 작업을 진행할 수 있었다. 초기에 약간의 어려움이 있었고 때로는 불가해한 지식도 필요했지만, 우리의 build 작업은 상징적, 실질적으로 결국에는 모든 것들이 작동하도록 만들었다.

## 잘못된 선택

**1. 플랫폼 균형.** 세 가지 플랫폼에 걸친 타이틀 발매를 해본 경험이 전무했기 때문에, 우리가 완전히 과소평가 내지 간과했던 여러 가지 세부 사항과 관련된 문제가 있었던 것은 어찌 보면 당연하다. QA 팀이 가동될 당시 그은 플레이스테이션 2에 몰두하고 있었다. QA 팀원 상당수가 예전에 플레이스테이션 2를 경험했다는 점에서는 행운이었다. 신속하게 좋은 리듬을 갖출 수 있었기 때문이다. 우리가 적시에 새로운 게임 버전을 제공하면 QA는 그것들을 버그 데이터베이스에 개별 엔트리로 세분

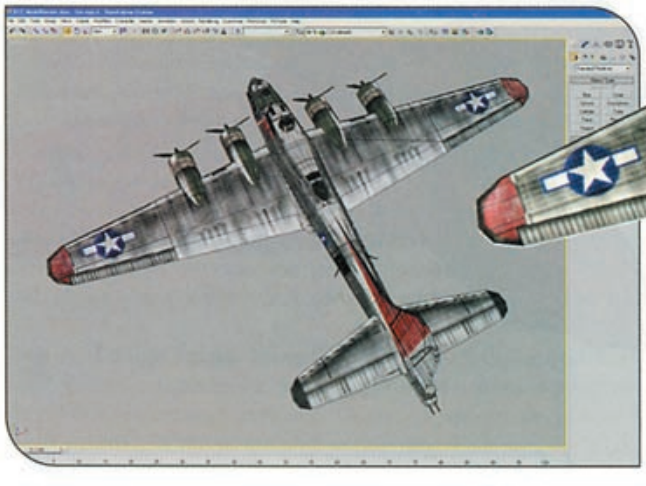
화시켜 짜 맞추었다. 플레이스테이션 2 버전을 안정화시킨 다음에는 X박스 버전으로 눈길을 돌렸다. 바로 이때 문제가 시작된 것이다. 플레이스테이션 2에 전적으로 초점을 맞추었기 때문에, X박스 버전은 타 버전에 비해 많은 기능들이 부족했다. 게다가 매우 불안정했다. 이 상황이 게임 개발에만 있는 것은 아니지만, 플레이스테이션 2 SKU를 최종 완성하는 단계였으므로 심각한 인력 부족에 직면했다.

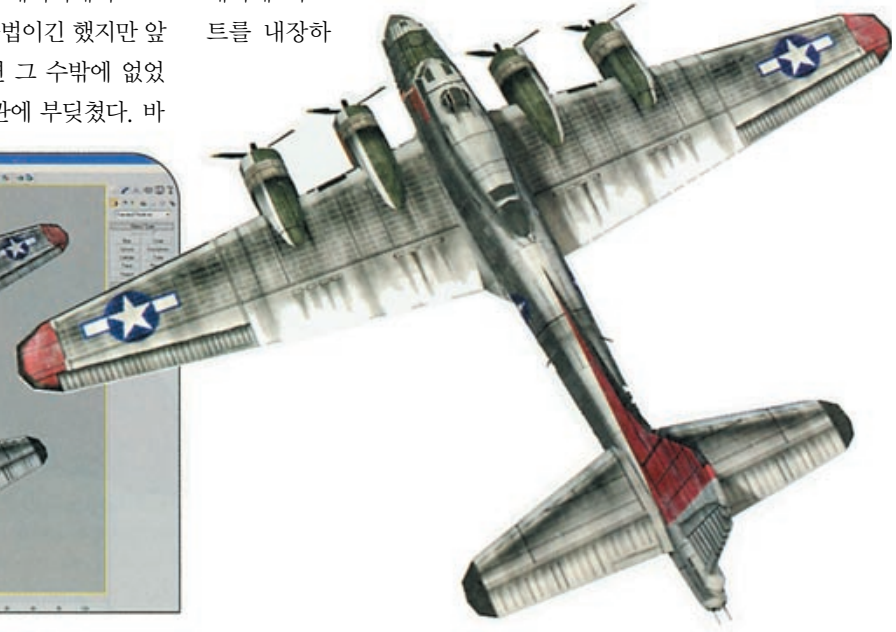
버그 치료를 담당한 인원들이 플레이스테이션 2의 문제를 더욱 중요하게 여기는 상황에서 X박스에서 버그가 발견됐다. 덕분에 작업 시간이 느려졌고 QA 팀이 X박스 버전을 면밀하게 분석할 시간은 줄었다. 이 상황은 우리가 최종 플레이스테이션 2 버전을 소니에 의뢰한 이후 다소 완화됐으며, 이때서야 모든 관심이 X박스에 집중됐다. 재빨리 X박스와 PC 버전을 납품 가능한 상태로 만들었지만, 이 모든 과정을 다시 한 번 거칠 기회가 있었더라면 버전의 우선순위를 조절해 각 버전마다 TLC 테스트 시간을 더 많이 부여했을 것이다. 진실은 단순하다. 테스트 시간을 더 많이 부여하면 더욱 많은 버그를 찾을 수 있고, 이들 버그를 신속히 처리하면 보다 깔끔하고 즐거운 완성품이 만들어지는 것이다.

**2. 인내.** 팀원 상당수가 여러 편의 타이틀에 참여한 경력이 있는 베테랑들이어서 위기의 순간마다 정신없는 상황에 익숙해져 있었다. 하지만 한 가지 플랫폼의 타이틀을 발매하는 것이 단거리 육상 경주라면 세 가지 플랫폼의 타이틀을 발매하는 것은 철인 3종 경기다. 플레이스테이션 2 버전의 경우 알파, 베타, 의뢰의 과정을 거쳐 오면서 기력이 떨어지기 시작했다. 하지만 X박스 버전 역시 똑같은 과정을 다시 거쳐야 했다. 남은 시간이 촉박했기 때문에 알파와 베타 간의 경계가 불분명해졌다. 정식 일정이 일일 스프레드시트와 공격적인 버그 데이터베이스 모니터링으로 대체됐다. 다소 두서없는 개발 접근법이긴 했지만 앞으로 처리해야 할 엄청난 작업량을 감당하려면 그 수밖에 없었다. 프로젝트 중 한 요소를 끝내자 또 다른 난관에 부딪쳤다. 바로 PC 버전이었다. 다른 두 버전과 마찬가지로 끝까지 지독한 레이스였으며, X박스와 PC가 매우 유사할 것이라는 우리의 가정은 여지없이 깨졌다. (대부분 하드웨어의 호환성과 관련된) 플랫폼 특유의 문제 외에도, PC 버전은 작업 완료 후 복제할 수 있도록 제 시간에 전달을 해야 했다.

지금 돌이켜보면 1차 플랫폼으로써 플레이스테이션 2에 주력했던 일은 결국 양날의 칼이 된 것이다. 덕분에 일반적으로 보다 까다로운 플랫폼에서의 개발 과정에서 부딪치게 되는 난관들을 처리할 수 있었고, 여타의 플랫폼보다 훨씬 일찍 선을 보일 수 있는 버전까지 얻게 됐다. 하지만 이러한 초점으로 인해 다른 두 버전이 너무 많이 뒤처지게 됐다. 보다 이상적인 개발 공정이었다면 모든 버전을 수평에 가까운 성숙도로 유지했을 것이다. 매달 각각의 플랫폼에서 어느 정도의 진전을 이루기는 했지만, 결과적으로는 제시간에 완성하기 위해 다른 플랫폼에 대한 포팅 기능을 서둘러 끝낸 것이다. 모든 버전이 제 시간에 완성됐고 품질 수준도 달성했지만, 팀이 치룬 대가는 의도했던 것 이상이었다. 가급적 앞으로의 일정은 모든 버전의 표준화에 초점을 두고 지표, 데모, 인적 자원 등을 염두에 둘 것이다.

**3. 국제화.** 세계 게임 시장이 성장하고 게임 개발비도 상승하면서, 가능한 한 많은 이들에게 게임을 선보이는 것이 중요해졌다. 우리의 경우 통상적으로 번역을 염두에 두었다. 그래픽에 텍스트를 내장하





지 않았고 UI 레이아웃 및 조정을 관리할 수 있도록 인터페이스를 꾸몄으며, 국제 정서에 맞게 스토리를 표현했다(우리의 게임은 2차 세계 대전을 배경으로 하고 있다는 이유로 독일과 일본 시장에서 비상한 관심을 모았다). 한 가지 간과했던 것은 텍스트 문자열 저장 방법과 그에 따른 게임 준비 파일로의 익스포트 방법이었다.

우리는 ID-태깅 문자열을 저장하는 데 맞춤형 테이블 기반 템플릿을 갖춘 MS 워드를 사용했다. 이 방법은 게임 디자이너들이 친숙하고 창조 친화적인 포맷으로 텍스트를 입력하는 데는 효과적이었다. 하지만 데이터 관리에는 적합하지 않은 것으로 드러났다. 초기에는 음성 기록 스크립트를 작성하는 과정에서 이것이 문제가 됐다. 기록을 위해 (개별 파일이 65개가 넘는) 여러 개의 문서를 하나의 포괄적인 MS 엑셀 스프레드시트로 잘라 붙여야 했다. 개별 문서를 채택해 하나의 포괄적인 데이터베이스로 만드는 자동화 시스템을 바로 이때 만들어야 했다. 하지만 우리는 그 순간에 얽매여 적절한 도구를 만드는 대신 이것을 손수 작업하는 등 현명하지 못한 결정을 내렸다.

게임 현지화 작업을 할 때 또 한 번 자동화된 텍스트 통합 프로세스의 부족을 통감해야 했다. 우리의 전체 텍스트 문자열은 최대 5,000개로, 비행 게임을 기준으로 봐도 그리 과도하게 크지 않은 것이 확실하다. 하지만 이 게임을 다섯 가지 언어로 현지화했다. 번역된 텍스트는 언어 고유의 세트로 돌아올 것이다. 곧 우리는 다섯 가지 번역된 언어 세트와 직면하게 됐는데, 각 세트 당 65개 이상의 문서가 있었고 모두 우리의 메인 데이터베이스로 통합시켜야 했다. 짧은 기간 동안 이것을 손수 처리하려는 시도가 있었다. 당연히 프로세스는 곧잘 오류로 이어지곤 했다. 전체 게임에 걸쳐 텍스트 불일치와 버전 제어 문제가 사방에서 돌출했다. 또한 손수 통합을 처리하는 데는 많은 시간이 소요되며, 앞으로 테스트할 신규 build를 만드는 데 들어가는 시간도 크게 지체된다. 마감 시간이 임박하면서, 우리는 자동화된 시스템을 만드는 문제를 정면으로 돌파해야 할 때가 됐음을 직감했다. 정식 개발에서 남은 시간이 두 달 밖에 없었다. 이러한 혼란을 겪은 결과 기술 팀장이 약 5시간에 걸쳐 자동화 된 문자열 임포터/익스포터를 만들게 됐다. 이 과정을 거친 덕분에 한때 (문서 세트 당)한 시간이 걸리던 것이 30분의 기적으로 바뀌게 됐다. 마침내 약 4시간 만에 DVD로 부팅을 할 수 있게 현지화 된 build 뱃치 전체를 변환할 수 있게 됐다. 독자들도 짐작할 수 있듯 QA 시간이 대폭 늘어나 깔끔하고 세련되며 잘 다듬어진 게임 버전을 해외로 공급할 수 있게 됐다.

이 이야기에서 얻을 수 있는 교훈은 바로 build 도구이다. 초반부터 시간이 많이 소모되면 나중에 가서 힘들어진다. 앞으로는 번역사가 웹을 통해 텍스트를 수정하면 우리의 build 프로세스로 자동 통합될 수 있는 온라인 데이터베이스 시스템을 구축할 계획이다.

**4. 미술 도구.** 이 요소를 "잘못된 선택" 목록에 넣게 된 것은 필자의 입장에서 하나의 고통이다. 우리는 훌륭한 도구를 만드는 데 주력했고 개발 기간 내내 그것을 지원했다. 적시에 뛰어난 도구를 구축하는 일의 중요성은 아무리 강조해도 지나치지 않다. 우리의 미술 도구는 마지막에 가서 성숙해지는 경우가 너무 잦다. 이번 프로젝트 중 상당 기간 원래의 플랫폼 환경(플레이스테이션 2 또는 X박스)에서 미술을 검토하는 데 애를 먹었다. 게다가 특수 효과 도구 역시 알파 두 달 전에야 완전히 온라인으로 연결됐다. 이 때문에 반복해서 다듬는 시간이 대폭 줄었다. 최종 게임 버전은 환상적이었지만, 프로젝트 초반부터 보다 강력하고 아티스트 친화적인 도구를 갖추고 있었더라면 특히 각 플랫폼의 잠재력 극대화라는 측면에서 훨씬 커다란 성취를 이룰 수 있었을 것이다. 프리프로덕션 기간 동안 여러 가지의 "강조 케이스"를 만들어 실제 개발에 필요한 기능 세트의 범위를 정의해야 했던 것이다.

**5. 제품 메시지 전달.** 공중전 시장은 복잡 미묘한 괴물이다. PC의 경우 하드코어 비행 시뮬레이션이나 우주 전쟁, 또는 "아케이드식" 슈팅 게임이 주류를 이룬다. 콘솔은 장르 정의라는 측면에서 이보다 약간 범위가 넓다. 토털리게임즈의 장르 역사는 축복인 동시에 저주이기도 하다. '독일 공군의 비밀 병기' 게임은 우리가 하드코어 비행 게임을 만들 준비를 하고 있다는 그릇된 인상을 심어줬다. 우리는 이러한 오해를 풀려고 노력했지만 어디서도 진정한 메시지가 팬들에게 전달되지 않았다. 놀랍게도 (토털리게임즈에게는 생소한) 콘솔 팬들이 우리의 입장을 수용하고 포용했지만, PC 플레이어들은 우리가 채택하기로 결정한 방향에 실망한 듯 보였다. 우리도 콘솔과 PC 게이머들 사이에 차이가 있다는 것을 이해하고 있었지만 게임의 메시지 전달 측면에서 제각기 매우 특정적인 취급을 요한다는 사실을 깨닫지 못했던 것이다. 플랫폼에 따라 게임이 수용되는 장르 라벨의 종류를 반드시 인식해야 한다. 돌이켜보면 커뮤니티 기반을 활성화시키고 이들에게 게임의 실체와 그것의 진행 과정을 꾸준히 전달해 주었다면 더욱 좋은 성과를 거두었을 것이다.

우리는 정보로 넘쳐나는 세계에 살고 있으며 매일같이 계속되는 개발이라는 도전에 몰두하기는 쉽지만, 이러한 최선의 노력이 성과로 이어질 수 있었던 것은 모두 우리의 팬들과 게임 커뮤니티의 덕분이다. 이들은 톡톡 튀는 아이디어의 귀중한 원천이며 영감의 근원지이기도 하다. 게임의 진흥 및 지원을 위한 현재 우리의 노력이 끈질긴 오해를 타파할 수 있을지 여부는 시간이 말해줄 것이다. 결국 여러 비평에서도 뒷받침하고 있듯, 우리는 재미있고 짜릿한 공중전 게임 제작이라는 목표를 달성했다.