http://dx.doi.org/10.5392/JKCA.2011.11.11.019

OpenGL을 이용한 Direct3D 기능의 구현

Implementing a set of Direct3D Functions on OpenGL

도주영*, 백낙훈**
경북대학교 대학원 컴퓨터공학과*, 경북대학교 IT대학 컴퓨터학부**

Joo-Young Do(djy@naver.com)*, Nakhoon Baek(oceancru@gmail.com)**

요약

본 논문에서는 윈도우(Windows) 기반의 데스크탑 환경에서, 특히 컴퓨터 게임에 관련된 응용 프로그램들에서 활발하게 사용되는 Direct3D가 제공하는 핵심적인 기능들과 함수들을 리눅스 환경에서 OpenGL 라이브러리를 기반으로 에뮬레이션 형태로 제공하고자 한다. 리눅스(Linux) 운영 체제 하에서는 일반적으로 X 윈도우 시스템과 OpenGL 라이브러리만 사용 가능한 컴퓨터 그래픽스 환경이 제공된다. 이 상황에서는, 우선적으로 윈도우 기반의 PC에서 Direct3D를 기반으로 개발된 게임 프로그램들이나 사용자 인터페이스 등을 컨버전하는 변환 작업에서 이러한 에뮬레이션 환경이 반드시 필요하다. 본 논문에서는 DirectX 9.0을 기준으로, 많이 사용되는 함수들을 선별하여 이들을 지원함으로써, 최종적인 전체 구현의 프로토타입을 확보하였다. 본 논문의 구현은 3차원 좌표 변환(3D coordinate transformation), 광원 및 재질(light & material), 텍스처 매핑(texture mapping), 애니메이션(animation) 등을 지원하여, 이를 이용하여 다양한 응용 프로그램 예제들과 실제 게임 캐릭터의 애니메니션 스크립트를 성공적으로 실행하여, 유용성을 입증하였다.

■ 중심어 : | DirectX | OpenGL | 그래픽스 표준 구현 |

Abstract

In this paper, we present an emulation library for the essential features and their API function calls provided by Direct3D, the most actively used API for computer game-related application programs on the MS-Windows-based desktop's, with OpenGL library in the Linux environment. In typical Linux-based systems, only the X window system and OpenGL graphics library are available. There are lots of needs for this kind of emulation library to convert the Direct3D-based game applications and user interfaces on these systems. Through carefully selecting the essential API functions from the DirectX version 9.0, we obtained the prototype implementation of that emulation library, to finally get the final full-scale DirectX implementation. Our implementation currently covers 3D coordinate transformations, light and material processing, texture mapping, simple animation features and more. We showed its feasibility through successfully executing a set of Direct3D demonstration programs including a real-world game character animation on our implementation.

■ keyword : | DirectX | OpenGL | Graphics Standard Implementation |

* 이 논문은 2011년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(과제번호: 2011-0014886).
접수번호 : #110830-001
접수일자 : 2011년 08월 30일
심사완료일 : 2011년 10월 07일
교신저자 : 백낙훈, e-mail : oceancru@gmail.com

I. 서 론

본 논문에서는 개인용 컴퓨터 시스템에서 운영 체제로 흔히 선택되는 리눅스 또는 임베디드(embedded) 리눅스 운영 체제에서 DirectX의 3차원 그래픽 기능인 Direct3D의 중요한 기능들을 지원하는 시스템을 구현하고자 한다. 리눅스 기반의 시스템들은 안드로이드를 비롯한 휴대폰 기반의 운영체제들의 기반이 되기도 하며, 많은 부분에서 호환성을 확보함으로써, 그 중요성이 더욱 커지고 있다[1-3].

Direct3D는 마이크로소프트 사의 윈도우 운영 체제를 사용하는 PC들에서는 게임이나 사용자 인터페이스 개발 시에 그래픽스 출력을 담당하는 가장 중요한 라이브러리로 사용되고 있다[4]. 반면에, 윈도우 운영 체제를 제외한 다른 시스템들에서는 그 폐쇄성 때문에, 사용하기가 상당히 곤란한 상황이기도 하다.

본 논문에서는 기존의 PC 기반 개발 환경들에서 DirectX를 기반으로 이미 개발되어 있는 그래픽스 및 게임 프로그램들을 새로 개발된 리눅스 기반의 소형 시스템에서 그대로 사용할 수 있는 환경을 구축하기 위한 첫 단계로, DirectX의 구성 요소들 중에서 특히 3차원 그래픽스 출력을 담당하는 Direct3D의 그래픽스 출력 함수들을 리눅스 기반 시스템에서 사용할 수 있도록 하는 에뮬레이션 방식의 그래픽스 라이브러리를 개발하고자 한다.

이는 기존의 PC 기반 개발 환경에서 개발된 그래픽스 및 게임 소프트웨어들을 그대로 사용할 수 있도록 함으로써, 게임 및 멀티미디어 콘텐츠 전반에 대한 비용 감소 효과를 가져 올 수 있을 것으로 기대된다.

결과적으로, [그림 1]의 왼쪽에서 보는 바와 같이, Direct3D 응용 프로그램이 Direct3D 라이브러리와 윈도우 운영 체제를 사용하여 출력하는 것과 동일한 결과를 본 논문에서 구현하는 에뮬레이션 라이브러리에서 OpenGL (또는 OpenGL ES) 라이브러리와 리눅스 운영 체제에서 얻을 수 있도록 하려는 것이 본 논문에서 지향하는 최종적인 목표이다.

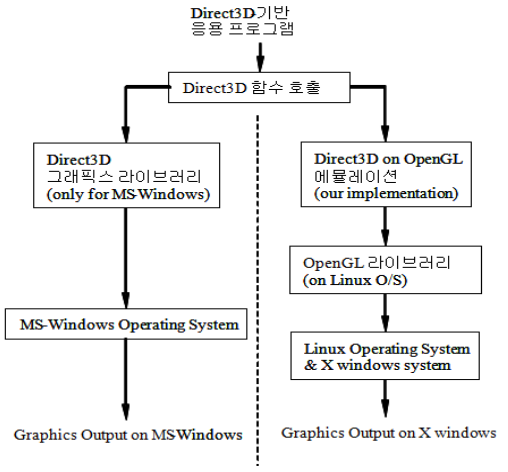


그림 1. 본 논문에서 제공하는 에뮬레이션 라이브러리의 역할

II. 기존 연구

본 논문에 직접적으로 관계된 기술들로는 OpenGL 라이브러리와 DirectX 개발 기술들을 들 수 있다. 이 장에서는 각각에 대해서 간략히 설명하겠다.

리눅스 시스템을 사용하거나, 특히 임베디드 시스템에서는 주로 OpenGL을 지원하였다. OpenGL [5]은 가장 광범위하게 쓰이고 있는 3차원 그래픽스 라이브러리 이며, 현재도 끊임없이 개량을 거치고 있다. 특히, 휴대폰을 비롯한 소형 임베디드 시스템을 위해 개발된, OpenGL ES(embedded system) [6-8]는 PC 등의 범용 기계에서 사용되던 3차원 그래픽스 라이브러리를 휴대폰을 비롯한 소형 임베디드 시스템으로 포팅할 때, 어떻게 처리해야 하는지를 보여주는 좋은 예제로 언급된다.

윈도우 운영체제에서는 OpenGL 대신 DirectDraw 또는 Direct 3D등의 그래픽스 라이브러리들이 별도로 제공되고 있다. 이들은 다른 기능들과 통합되어 DirectX라는 이름으로 제공되고 있는데, 특히 윈도우 환경에서는 가장 널리 쓰이고 있다.

비주얼 스튜디오(Visual Studio) 에서는 임베디드 시스템에 대한 꾸준한 지원을 해 왔고, 이에 따라, 임베디드 시스템용 비주얼 스튜디오(Visual Studio for Embedded System) 등이 제공되었다. 이들은 내부에

윈도우 CE 에뮬레이터 등이 제공되어, PC를 임베디드 시스템 개발 환경으로 사용 가능하다. 비주얼 스튜디오 닷넷에서는 다양한 통합을 시도하였으며, 그 중의 하나가 임베디드 시스템 개발 환경이 완전히 합쳐진 것이다. Visual Studio .NET 2005 이후로는 PC, 임베디드 시스템 모두가 사용 가능하다.

Ⅲ. 설계 및 구현

본 논문에서 구현한 에뮬레이션 라이브러리의 구현에 있어서는 우선 기존의 Direct3D 기능들 중에서 어디까지를 구현할 것인지 부터 결정하여야 할 것이다. 본 논문에서는 현재 국내의 게임 업계에서 선호되는 버전(version)을 기준으로, Direct3D 버전 9을 대상으로 선정하였다. 또한, 기능 면에 있어서는 GPU를 직접 이용하는 버텍스 쉐이더(vertex shader)나 픽셀 쉐이더(pixel shader) 기능을 시뮬레이션하기 위해서는 버텍스 쉐이더 또는 픽셀 쉐이더가 필요로 하는 내부 컴파일러 등의 시스템을 완전히 새롭게 구현해야 하므로, 본 논문의 범위를 벗어난다. 이에 따라, Direct3D 버전 9 중에서도 주로 FVF(Flexible Vertex Flags)를 사용하는 그래픽스 프로그래밍 방식을 대상으로 하였다.

이에 따라, 구현의 대상이 되는 Direct3D 클래스들은 다음과 같다.

- **IDirect3D9** : Direct3D 이용의 기본이 되며, 여기서 IDirect3DDevice9 클래스를 생성하는 방식을 취한다.
- **IDirect3DDevice9** : Direct3D 이용의 핵심이 되는 클래스이며, 그래픽스 프리미티브 처리 등의 핵심 기능이 이 클래스에서 처리된다.
- **IDirect3DTexture9** : 텍스처 매핑(texture mapping) 기능을 지원하기 위한 추가 클래스.
- **IDirect3DVertexBuffer9** : 그래픽스 프리미티브(graphics primitive)에 좌표 정보를 주기 위한 클래스.
- **D3DXMATRIX** : 행렬 처리를 위해 추가된 클래스.

위의 클래스들을 구현하는 과정에서, 다음과 같은 구조체(structure)들이 부수적으로 필요하며, 이들도 함께 제공된다.

- **_D3DCOLORVALUE** : (R, G, B, A)로 구성된 4개 channel 컬러 정보를 제공한다.
- **_D3DVECTOR** : (x, y, z)로 구성된 3차원 벡터를 제공한다.
- **_D3DRECT**,
- **_RGNDATA**,
- **_RGNDATAHEADER** : 모두 화면 상의 일정 영역을 의미하는 영역(region)을 정의한다.
- **_D3DLIGHT9** : 광원을 처리하기 위한 라이트(light)를 정의하는 기능을 제공한다.
- **_D3DMATERIAL9** : 광원 처리 과정에서 물체의 특성인 재질(material)을 정의하는 기능을 제공한다.
- **_D3DPRESENT_PARAMETERS** : 전체 화면의 업데이트(update)에 관계된 기능들을 제공한다.

핵심이 되는 클래스 별로 각각의 구현 함수들을 살펴보면, 우선 **IDirect3D9** 클래스에서는 다음의 함수들이 제공된다.

- *IDirect3D9* (void)
- *~IDirect3D9* (void)
- *Release* (void)
- *CreateDevice* (...)

다음으로, **IDirect3DDevice9** 클래스에서는 다음 함수들이 제공된다.

- *IDirect3DDevice9* (...)
- *~IDirect3DDevice9* (void)
- Release (void)
- CreateVertexBuffer (...)
- BeginScene (void)
- EndScene (void)
- Clear (...)
- SetStreamSource (...)
- SetFVF (DWORD fvf)
- DrawPrimitive (...)
- Present (...)

- SetTransform (...)
- SetRenderState (...)
- SetTexture (...)
- SetLight (...)
- LightEnable (...)
- SetMaterial (...)

텍스처 매핑을 지원하기 위한 **IDirect3DTexture9** 클래스에서는 꼭 필요한 함수만 지원하여, 다음과 같은 함수들이 지원된다.

- IDirect3DTexture9 (void)
- ~IDirect3DTexture9 (void)
- void FromRGB (...)
- void FromBGR (...)
- void FromBGRA (...)

그래픽스 프리미티브(graphics primtive)들에서 필요로 하는 **IDirect3DVertexBuffer9** 클래스에서도 꼭 필요한 기능만 구현하여, 다음 기능들이 제공된다.

- IDirect3DVertexBuffer9 (...)
- ~IDirect3DVertexBuffer9 (void)
- ULONG Release (void)
- HRESULT Lock (...)
- HRESULT Unlock (void)

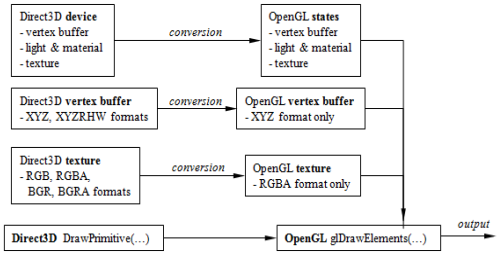


그림 2. 전체적인 소프트웨어 구조

본 논문에서 구현한 함수들은 전체적으로는 [그림 2]에서와 같은 소프트웨어 구조를 가지도록 하였다. 전체 소프트웨어의 실제 출력은 Direct3D의 DrawPrimitive( ) 함수가 담당하고, 이 함수는 내부적으로는 OpenGL의 glDrawElements( ) 함수로 변경되어 실제 출력을 수행한다. Direct3D의 device, vertex buffer, texture들은 미리 대응되는 OpenGL 용의 state variable, vertex buffer, texture로 변환되어져 있도록 하여 전체 효율을 높였다.

실제 구현에서는 Direct3D와 OpenGL 간의 설정 방식에서 오는 차이의 정도가 어느 정도인지에 따라 세부 구현이 달라지는 방식을 택하였다. 우선, 기능 상의 차이는 별로 없고, 단순한 문법 상의 차이에 대해서는 비교적 단순하게 해당되는 설정값들을 찾아내어, 정확히 대응되는 값들을 설정하도록 하였다. 예를 들어, 3차원 그래픽스 파이프라인에서 깊이 버퍼(depth buffer)를 사용할 것인지 않을 것인지의 설정은 Direct3D에서는 각각 다음과 같이 설정한다.

```
SetRenderState(D3DRS_ZENABLE, D3DZB_TRUE);
SetRenderState(D3DRS_ZENABLE, D3DZB_FALSE);
```

대응되는 OpenGL 설정은 각각 다음과 같다.

```
glEnable(GL_DEPTH_TEST); // z-buffer enabled
glDisable(GL_DEPTH_TEST); // z-buffer disabled
```

본 논문에서는 이런 경우들에 대해서는 대응되는 값들이 정확히 설정되도록 각각의 사례들을 모두 넣어두는 방법을 택하여, 다음과 같이 처리하였다.

```
IDirect3DDevice::SetRenderState(D3DRENDERSTATETYPE state, DWORD value) {
    // pre-processing
    ...
    // main processing
    ... /* other cases */
    if (state == D3DRS_ZENABLE) {
      if (value == D3DZB_TRUE) {
        glEnable(GL_DEPTH_TEST);
      } else {
        glDisable(GL_DEPTH_TEST);
      }
  }
    ... /* other cases */
```

```
    // post-processing
    ...
}
```

이런 방식의 처리 사례는 실제로는 드물게 발생하고, 대부분의 경우는 설정 방식이 서로 다르거나, 매개변수의 변환이 필요한 경우가 절대 다수를 차지한다. 예를 들어, 광원의 계산에서 반드시 필요한 주변광(ambient light)의 설정에 대해서는 Direct3D에서는

```
SetRenderState( D3DRS_AMBIENT, 0x800080FF );
    // 0x800080FF is an RGBA value
```

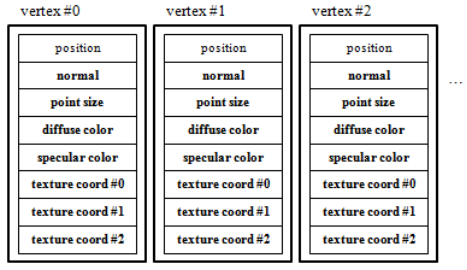
와 같은 방식을 쓰지만, OpenGL 에서는 대응되는 설정을 다음과 같이 해야 한다.

```
float value = { 0.5F, 0.0F, 0.5F, 1.0F };
glLightModelfv(  GL_LIGHT_MODEL_AMBIENT, value );
```

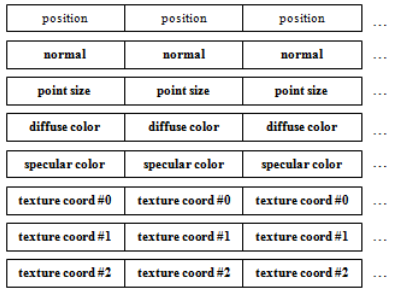
본 논문에서는 두 그래픽스 표준 사이의 대응 관계를 명확히 파악하여, 이러한 설정 방식의 차이와 매개변수의 변환을 모두 처리하도록 하였다. 다음은 위의 대응 관계를 처리하는 실제 코드를 간략히 제시한 예이다. 실제 코드에서는 갖가지 에러 상황들과 예외 처리를 위해서, 상당한 전처리와 후처리 과정을 필요로 한다.

```
IDirect3DDevice::SetRenderState(D3DRENDERSTATETYPE state, DWORD value) {
    // pre-processing
    ...
    // main processing
    ... /* other cases */
    if ( state == D3DRS_AMBIENT ) {
        value[0] = RGBA_GETRED(value) / 255.0F;
        value[1] = RGBA_GETGREEN(value) / 255.0F;
        value[2] = RGBA_GETBLUE(value) / 255.0F;
        value[3] = RGBA_GETALPHA(value) / 255.0F;
        glLightModelfv(
            GL_LIGHT_MODEL_AMBIENT, value );
    }
    ... /* other cases */
    // post-processing followed
    ...
}
```

그래픽스 프리미티브의 실제 출력을 담당하는 DrawPrimitive 함수의 경우는 더욱 복잡한 처리를 필요로 하는데, 근본적으로 Direct3D에서는 [그림 3.(a)]에서와 같이, FVF (flexbile vertex flags) 라는 개념에 따라, 꼭지점 하나에 필요한 모든 정보를 하나의 구조체에 모으는 방식을 택하였지만, OpenGL에서는 [그림 3.(b)]에서와 같이, 필요한 정보들마다 별도의 배열을 부여하는 방식을 택하고 있다. 따라서, 우리의 구현에서는 Direct3D 용으로 들어오는 버텍스 정보를 분석하여, 각 정보 별로 별도의 버퍼들로 완전히 분리시키고, 이들 버퍼 별로 필요한 기능들을 설정하여, 원하는 출력이 나오도록 하였다. 아래 예제는 DrawPrimitive의 내부에서 포지션(position) 부분의 처리만을 요약한 것이다.



(a) Direct3D의 FVF 저장 방식



(b) OpenGL에서의 저장 방식

그림 3. 꼭지점(vertex) 저장 방식의 차이

```
IDirect3DDevice9::DrawPrimitive(
  D3DPRIMITIVETYPE primitiveType,
  UINT startVertex, UINT primitiveCount) {
  // pre-processing
  ...
// main processing
  // position values
  ...
  switch (m_fvf & D3DFVF_POSITION_MASK) {
  case D3DFVF_XYZ:  // (x, y, z) 형식
    ... (해당 처리 후, vertex buffer 설정)
    break;
  case D3DFVF_XYZW: // (x, y, z, w) 형식
    ... (해당 처리 후, vertex buffer 설정)
    break;
  case D3DFVF_XYZRHW: // (x, y, z, 1/w) 형식
    ... (해당 처리 후, vertex buffer 설정)
    break;
  case D3DFVF_XYZB1:
    ... (해당 처리 후, vertex buffer 설정)
    break;
    ... (기타 경우는 생략)...
  }
  ...
  // normal values
  ... (omitted)
  // point size values
  ... (omitted)
  // diffuse values
  ... (omitted)
  // specualr values
  ... (omitted)
  // texture coordinate values
  ... (omitted)
  // post-processing followed
  ...
}
```

이 부분은 상당히 복잡하면서도 정확한 처리가 필요한 부분이었고, 다음 장에 나오는 예제 프로그램들을 통하여 사례 별로 테스트 과정을 거쳐, 정확한 구현이 가능하였다. 다음 장에서는 사례 별로 이들이 처리되는 예제들을 보이겠다.

Ⅳ. 구현 결과

개발한 그래픽스 라이브러리가 기존의 Direct3D와 호환됨을 보이기 위해, 다양한 데몬스트레이션 프로그램들을 개발하여, 그 결과를 비교하였다. 이들에 대해서 하나씩 설명하고, 각각의 단계에서 해결해야 했던 기술적 문제들을 설명하겠다.

[그림 4]의 프로그램은 윈도우 환경에서 정상적으로 작동하는 Direct3D 프로그램이 리눅스 환경에서도 작동할 수 있음을 보이려는 목적으로 제작되었다. 프로그램 자체는 파란(blue) 바탕 화면에 흰색(white) 삼각형을 출력하는 간단한 것이다.

반면에, 프로그램 내의 기술적인 부분들을 살펴본다면, 윈도우 환경과 리눅스에서 모두 돌아가는 일관된 윈도우 관리 기법을 완성하였다는 의의를 가진다. 본 논문에서는 양쪽 모두에서 작동하는 그래픽스 프로그램을 작성하는 방법으로 GLUT [9]를 사용하는 기법을 사용하였다. 이 기법은 위의 예제에서 보는 바와 같이, 윈도우에서 작성한 소스 코드(source code)를 일체의 수정 없이 사용할 수 있음이 증명되었다.

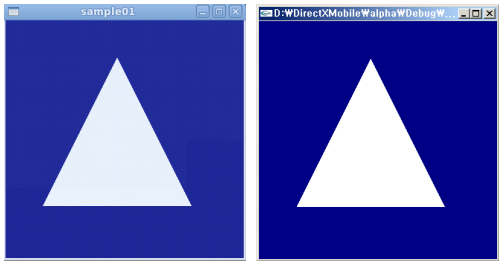


그림 4. 단순 삼각형의 출력과 윈도우 시스템 테스트 (왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

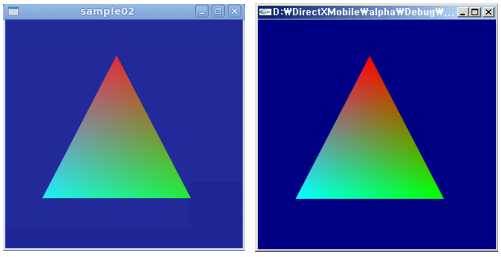


그림 5. RGBA 설정에 따른 컬러 테스트
(왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

[그림 5]는 광원과 재질 기능의 테스트에 쓰였던 프로그램으로, 역시 왼쪽의 구현 결과와 오른쪽의 윈도우 출력에서 차이점이 없음을 알 수 있다. 이 예제에서는 D3DFVF_XYZRHW의 지원, Direct3D 및 OpenGL의 color 해석 차이와 같은 기술적 문제들을 해결하였다.

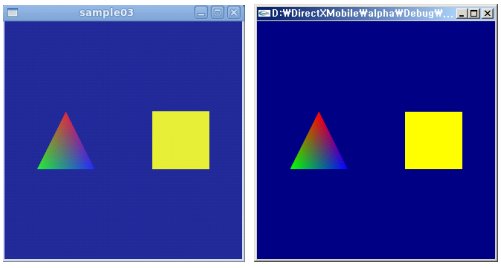


그림 6. 다양한 프리미티브 테스트
(왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

[그림 6]에서는 서로 다른 그래픽스 프리미티브(graphics primitive)들을 동시에 여러 개 출력했을 때도 문제가 없는 지를 체크하는 데에 주목적이 있었다. 실제 출력에 있어서는 이를 처리하기 위해, 행렬의 처리 방법을 구현하여야 한다.

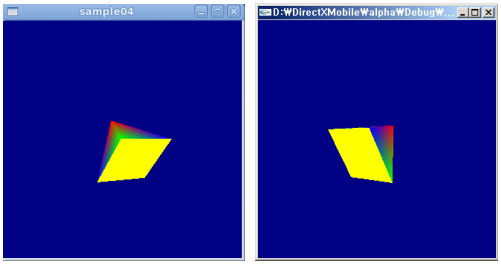


그림 7. 애니메이션 테스트
(왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

[그림 7]에서는 본격적인 애니메이션(animation)이 가능한가를 검사하였다. 이를 위해서, 사각뿔을 만든 후, 이를 실시간으로 회전시키는 예제를 작성하였다. 그림에서는 화면 1개만 제시하였지만, 실제 프로그램은 사각뿔을 계속해서 회전시키는, 동적 애니메이션을 보여준다.

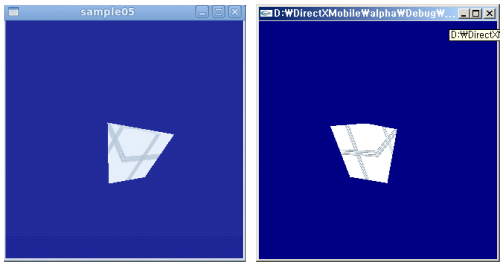


그림 8. 텍스처를 적용한 애니메이션
(왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

[그림 8]에서는 고급 그래픽스 프로그램들에서 반드시 필요한 텍스처 매핑(texture mapping) 기법을 구현하는 데에 초점을 맞추었다. 여기서는 특히 Direct3D와 OpenGL에서의 텍스처 매핑 처리 방법이 완전히 달라서, 몇가지 기술적 난점을 해결해야 했다.

Direct3D에서는 IDirect3DTexture9 클래스가 중요한 역할을 해서, CreateTextureFromFile( ) 함수에서 이미 모든 처리를 끝내고, SetTexture( ) 함수는 단순히 텍스처를 바인딩하는 역할만 수행한다. 반면에, OpenGL에서는 glTexImage2D( )와 같은 함수가 매우 많은 자원을 사용하는 함수이므로, 이에 대한 처리는 실행 회수를 제한시켜야 한다. 우리의 구현에서는 CreateTexture( ) 함수에서, 미리 텍스처를 모두 만들고, 그 이후는 SetTexture( ) 함수에서는 glBindTexture( )로 이미 만든 텍스처의 단순 바인딩만 수행하도록 하여, 실제 사용하는 자원을 최소화하였다.

[그림 9]에서는 고급 그래픽스 기법들에서 반드시 필요한 광원과 재질(L&M, lighting and material) 기능을 점검하였다. 실세계의 광원을 구현하기 위해서는 광원(lighting) 기능이 필수적이고, 물체들에서는 주어진 라이트에 대해서 반사를 행하는 정도를 구현하기 위한 재질(material) 기능이 필요하다. 특히 문제가 된 것은

Direct3D와 OpenGL에서의 광원과 재질 기능에 대한 처리가 완전히 다르다는 점이다.

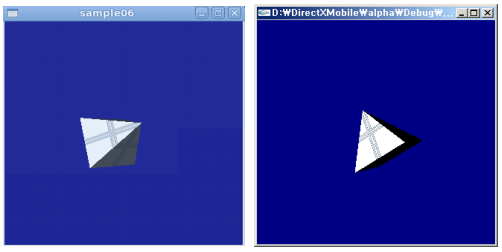


그림 9. 광원과 텍스처를 동시에 적용한 애니메이션 (왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

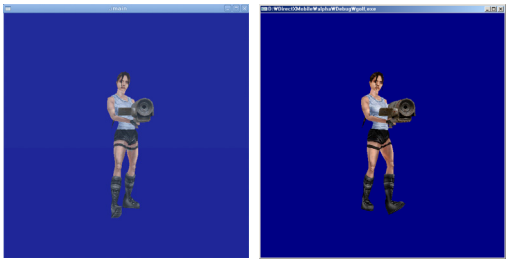


그림 10. 게임 캐릭터의 애니메이션 (왼쪽: 구현 결과, 오른쪽: Direct3D의 출력)

앞에서 언급한 프로그램들을 단계적으로 사용하여, 윈도우에서 구현한 Direct3D 프로그램이 일체의 수정 없이 리눅스 시스템에서 실행 가능함을 보였다. 이제 최종적으로 전체 구현의 결과를 보여 주기 위해, 사람과 흡사한 게임 캐릭터(character)를 이용하여, 정해진 애니메이션(animation)을 행하는 예제 프로그램을 구현하였다. 구현된 프로그램은 Direct3D로 구현되어, [그림 10]에서 보는 바와 같이, 윈도우 운영체제에서는 물론이고, 리눅스 시스템에서 OpenGL 기반으로 완전히 새로 구현된 본 논문의 에뮬레이션 라이브러리를 이용해서도 똑같이 실행가능하다. 이러한 프로그램들에서의 테스트를 통하여, 최종적으로 우리의 구현 결과가 잘 작동함을 보였다. 수행 속도 면에서는 윈도우즈 시스템과 리눅스 시스템의 상이한 수행 환경으로 인해, 단순 비교가 어려우나, 테스트에 사용한 모든 예제들에서 양쪽 모두 실시간 처리가 가능하였다.

V. 결론

Direct3D 응용 프로그램이 Direct3D 라이브러리와 윈도우 운영 체제를 사용하여 출력하는 것과 동일한 결과를 OpenGL (또는 OpenGL ES) 라이브러리와 리눅스 운영 체제에서 얻을 수 있도록 하려는 것이 본 논문의 목표였다.

이러한 Direct3D 클래스들을 사용하면, 기본적인 수준의 Direct3D 응용 프로그램들은 물론이고, 간단한 애니메이션 프로그램들까지 지원할 수 있음이 확인되었다. 특히, 이들을 검증하기 위한 응용 프로그램들이 다수 제작되었으며, 이를 통하여, 최종적으로 게임 캐릭터 애니메이션(game character animation)까지 가능함을 보였다.

다음 단계에서는 더 다양한 함수들을 지원하여, 프로토타입 시스템을 완성할 예정이다. 우리는 또한 이에 연관된 다른 그래픽스 라이브러리들, 즉, OpenGL ES 나 OpenVG 등의 새로운 그래픽스 라이브러리들도 구현하고 있으며, 이들은 이미 상업적인 서비스를 하고 있기도 하다.

참 고 문 헌

[1] K. Pulli, T. Aarnio, K. Roimela, and J. Vaarala, “Designing graphics programming interfaces for mobile devices,” IEEE CG&A, Vol.25, No.6, pp.66-75, 2005.

[2] http://code.google.com/android/what-is-android.html

[3] http://developer.apple.com/iphone/

[4] Y. H. Mirza and H. da Costa, “Introducing the New Managed Direct3D Graphics API in the .NET Framework,” *MSDN magazine*, 2003.

[5] M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification*, version 4.1, 2011.

[6] A. Munshi and J. Leech, *OpenGL ES Common/Common-Lite Profile Specification*,

version 1.1.12 (Full Specification), Khronos Group, 2008.

[7] A. Munshi and J. Leech, *OpenGL ES Common Profile Specification*, version 2.0.24 (full specification), Khronos Group, 2009.

[8] K. Pulli, T. Aarnio, V. Miettinen, K. Roimela, and J. Vaarala, *Mobile 3D Graphics: with OpenGL ES and M3G*, Morgan Kaufman, 2007.

[9] M. J. Kilgard, *The OpenGL Utility Toolkit (GLUT) Programming Interface*, version3, Silicon Graphics Inc., 1996.

저 자 소 개

도 주 영(Joo-Young Do) 정회원

- 1999년 2월 : 경성대학교 전산통계학과(이학사)
- 2001년 2월 : 경성대학교 컴퓨터과학과(이학석사)
- 2001년 3월 ~ 현재 : 경북대학교 컴퓨터학부 박사과정(공학박사 수료)

<관심분야> : 컴퓨터 그래픽스, 멀티미디어, 물리기반 렌더링

백 낙 훈(Nakhoon Baek) 정회원

- 1992년 2월 : 한국과학기술원 전산학과(공학석사)
- 1997년 2월 : 한국과학기술원 전산학과(공학박사)
- 2004년 3월 ~ 현재 : 경북대학교 컴퓨터학부 교수

<관심분야> : 모바일 그래픽스, 리얼타임 그래픽스