3D 영상 효과를 위한 레이어 채널 이미지의 처리 기법

Processing Techniques of Layer Channel Image for 3D Image Effects

최학현*, 김정희**, 이명학***
서울여자대학교*, 성균관대학교**, 숭실대학교***

Hak-Hyun Choi(choiidea@empal.com)*, Jung-Hee Kim(jhjasmin@yahoo.co.kr)**,
Myung-Hak Lee(medherbs@naver.com)***

요약

3D 영상에 이펙트를 표현할 수 있는 레이어 채널을 삽입함으로써 애플리케이션 렌더링에 효과적으로 이용하도록 한다. 현재의 이펙트 렌더링은 영상과 이펙트의 개별 처리 및 혼합의 방식을 사용하기 때문에 저장 공간과 영상 처리에 있어서 개별 소스를 필요로 하고 있다. 그러나 영상과 레이어 채널을 하나로 묶어 처리함으로써 비용 절약과 영상 처리 면에서 큰 효과를 볼 수 있다. 개발은 영상에 레이어 채널을 삽입하기 위해서 영상 포맷의 변경, 레이어 채널이 나타나지 않도록 숨김 기능 추가, 영상 로드시 영상과 레이어 채널을 동시 접근 가능하도록 제어, 영상과 레이어 채널이 쉽게 혼합될 수 있도록 간편한 알파 블렌딩 처리 등의 방법으로 영상 포맷을 변경하여 레이어 채널을 숨기는 기법, 일반 영상 뷰어에서도 변경된 포맷의 영상을 볼 수 있도록 개발, 레이어 채널과 영상을 같이 묶음으로써 재사용성을 높이고 모든 프로그램에 이용 가능하도록 만든다. 그러면 영상 로드시에 영상과 레이어 채널을 동시에 불러드림으로써 처리 속도 향상시키고 3D 영상에 레이어 채널을 삽입함으로써 레이어 채널 영상을 위한 소스 저장 공간을 줄일 수 있다. 또한 3D 영상과 레이어 채널의 영상을 한 번에 다룰 수 있게 되어 효과적인 이펙트 표현 가능하고 실제 애플리케이션이 될 수 있는 멀티미디어 영상 등에 효과적으로 이용이 가능할 수 있을 것으로 기대한다.

■ 중심어 : | 레이어 채널 | 영상 변환 | 레이어 혼합 | 이펙트 렌더링 | 알파 블렌딩 |

Abstract

A layer channel, which can express effects on 3D image, is inserted to use it on application rendering effectively. The current method of effect rendering requires individual sources in storage and image processing, because it uses individual and mixed management of images and effects. However, we can save costs and improve results in images processing by processing both image and layer channels together. By changing image format to insert a layer channel in image and adding a hide function to conceal the layer channel and control to make it possible to approach image and layer channels simultaneously during loading image and techniques hiding the layer channel by changing image format with simple techniques, like alpha blending, etc., it is developed to improve reusability and be able to be used in all programs by combining the layer channel and image together, so that images in changed format can be viewed in general image viewers. With the configuration, we can improve processing speed by introducing image and layer channels simultaneously during loading images, and reduce the size of source storage space for layer channel images by inserting a layer channel in 3D images. Also, it allows managing images in 3D image and layer channels simultaneously, enabling effective expressions, and we can expect to use it effectively in multimedia image used in practical applications.

■ keyword : | Layer Channel | Image Change | Layer Mixing | Effect Rendering | Alpha Blending |

접수번호 : #071214-006
접수일자 : 2007년 12월 14일
심사완료일 : 2008년 01월 03일
교신저자 : 최학현, e-mail : choiidea@empal.com

I. 서 론

많은 멀티미디어 영상 개발자나 연구원들이 특수효과를 이용하기 위해서 많은 양의 이미지 데이터를 혼합하거나 수식계산을 해서 다른 형태의 원하는 이미지 데이터를 얻어내고 있다. 그런 과정에서 문제점은 디지털 영상이라는 특수성인데, 메모리의 제한과 CPU의 연산처리와 그래픽하드웨어의 처리 속도의 한계점을 가지고 있다. 이러한 부담감을 갖고 디지털영상을 만들다 보니 디지털영상의 질적인 부분과 그래픽 처리의 속도라는 점을 신경 쓰지 않고 디지털영상을 제작할 수 없게 된다. 특히 그래픽 관련 연산은 컴퓨터에게 상당한 연산을 요구하는 것으로 많은 연산을 할수록 훌륭한 특수효과를 많이 연출할 수 있지만 그 만큼의 그래픽 처리의 속도가 현저하게 저하된다. 결국 디지털영상은 적당한 수준에서 만족해야 하고 디지털영상을 즐기는 플레이어들은 하드웨어를 업그레이드하지 않는 이상 훌륭한 특수효과를 보기 힘들다.

기존의 이미지는 단지 이미지 자체가 특별한 역할을 하는 것은 아니고 단지 데이터로서의 기능제공만 하고 있다. 단지 RGB값만 들어 있는 데이터들을 GIF그림 파일처럼 자체 애니메이션 기능이 있는 것과 같은 기능성이 있는 이미지 데이터로 가공하여 처리하는 개발 방법을 모색했다. 이미지 자체가 특수 효과를 처리할 수 있는 정보로 이루어진 데이터라면 간단한 이미지 특수효과를 어렵게 구현하지 않아도 되고 많은 양의 이미지 데이터도 필요 없게 되며 또한 특수 효과를 표현하기 위한 복잡하고 다양한 계산을 하지 않아도 된다. 디지털 영상안에서 표현되고 있는 특수효과를 위한 3D 영상의 레이어 채널 이미지 처리 기법 및 외부 배경 처리 속도 향상을 위해 레이어 채널 이미지 처리 기법을 개발하고자 한다.

본 논문에서는 레이어 채널 및 이미지 구조에 대해 고찰하고, 레이어 채널 이미지 처리를 연구한 후, 레이어 채널 고속 이미지 출력 기법 개발에 대해 실제 구현한 결과들을 소개하면서 본 연구의 결론과 함께 개발 효과들을 제시한다.

II. 레이어 채널 구조

1. 레이어 채널 이미지 파일

붉은색, 녹색, 파란색, 알파 값을 하나의 색의 정보 값으로 이루어진 데이터의 집합이다. 이런 데이터들은 [그림 1]에서 보는 바와 같이 시스템 메모리에서 약간의 가공을 하거나, 바로 비디오 메모리로 전송이 되어 최종 디스플레이 화면에서 출력이 된다. 본 기술개발의 초점은 이 과정에서 이미지 데이터의 원본을 수정하여 RGBA로 이루어진 데이터 안에 레이어 채널을 만들고 또 하나의 이미지 정보를 저장하여 원본이미지와 레이어 채널에 가공된 이미지인, 두 이미지를 하나의 데이터 구조에 넣고 이것을 가능하게 하여 기술 개발의 목적을 이뤘다.

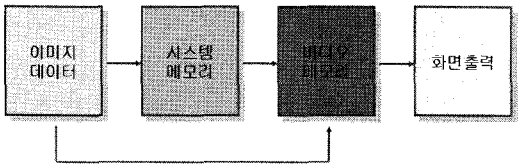


그림 1. 이미지의 처리 과정

기존의 이미지 파일들은 [그림 2]에서 보는 것처럼 헤더와 이미지 데이터, 팔레트 데이터의 구조를 가진다. 헤더의 역할은 이 이미지의 특성과 이미지의 사이즈, 압축 사용 여부 등의 정보가 들어있고 실제 눈으로 보일 이미지 데이터는 그 다음부터 나열된다. 마지막으로 팔레트라는 정보를 이용하여 적은 양의 데이터 표현으로 보다 많은 색을 표현해 줄 수 있는 정보를 이루어 하나의 이미지를 형성한다. 이 파일구조를 응용하여 새로운 이미지 파일 형태로 구현할 수 있는 방법으로 대체하였다.

그림 2. 이미지 파일의 기본 구조

2. 레이어 채널

디지털영상에선 [그림 3]에서 보는 것처럼 레이어라는 개념을 자주 이용한다. 표현하고자 하는 디지털영상에 [그림 3](b)에서처럼 레이어라는 계층을 이용하여 반복되는 디지털영상이 서로 다른 것처럼 일부에 변형을 주고자 하는 기법들이 자주 사용하기 때문이다. 그러나 [그림 3]에서 보는 것처럼 레이어는 3D 그래픽에서 물체나 지형 혹은 텍스쳐(Texture. 3D 그래픽에 사용되는 이미지를 지칭한다)의 표면을 한 개에서 여러 개를 겹겹이 쌓는 것이고 2D 그래픽에서는 스프라이트나 배경이미지에 또 하나의 그림을 겹치는 것이다. 결국 또 다른 이미지 데이터가 필요하고 서피스(그림 이미지 데이터를 컴퓨터 메모리의 어느 한 부분에 영역을 주어 관리할 수 있도록 하는 것)가 따로 생성되어야 하며, 또 따로 데이터를 로딩(컴퓨터 하드 드라이브 등의 보조기억장치에서 주기억장치인 메모리로 옮기는 과정을 지칭해준 것으로 주로 그림데이터나 음악데이터 등을 메모리상에서 관리하고자 하는 목적으로 생성하는 것을 의미한다)한다.

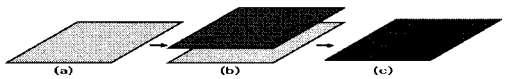


그림 3. 레이어 채널을 만든 이미지

3. 레이어 채널 이미지의 팔레트

32비트의 정보로 이루어진 이미지 데이터를 16비트로 떨어뜨리면 색의 질이 떨어지게 된다. 즉 32비트의 이미지란 2의 32승으로 4,294,967,296 가지의 색을 표현할 수 있지만 16비트 이미지는 2의16승인 65536 개의 색 이상은 표현할 수 없다는 의미이다. 이것은 4,294,901,760 가지의 색을 표현 할 수 없게 되는 현상을 초래하게 된다. 이것을 해결하기 위해서 팔레트를 사용하는 데 이 기법은 8비트 이미지가 256 개의 정보만 가지고 16비트 혹은 32비트처럼 그림을 표현 하려고 했던 의도에서 이미 만들어졌던 기법이다. 이것을 바로 레이어 채널 이미지에 적용하여 16비트 칼라를 갖는 이미지 팔레트 값을 갖고 [그림 4]와 같은 파일 구조를 가진다. 즉 32비트 칼라 값 중에서 실제로 이 이미지 한 장이 가지고 있는 색의 종류 중 65536개만 골라서 마치 32비트의 이미지처럼 표현하였다.

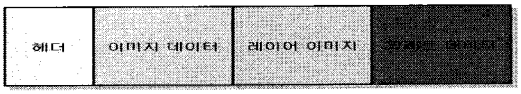


그림 4. 팔레트가 있는 레이어 채널 이미지 파일 구조

이 방법을 사용하기 위해서 기본적으로 아무런 이미지 데이터가 없어도 32비트의 색 중 65536가지의 색의 정보가 들어 있는 데이터 영역이 존재하게 되는데 그 크기는 65536 x 4 Byte(컴퓨터의 정보 표현 단위로 2진수 8자릿수 즉 8비트가 모이면 1바이트가 된다)로 262,144Byte의 공간이 필요로 하게 된다. 32 비트 이미지로 치면 가로 세로 256 x 256 Pixel(이미지의 해상도를 의미하는 것을 1 Pixel은 이미지의 한 점을 의미한다) 크기의 이미지가 된다. 이 팔레트 기능을 사용하여 32비트 이미지의 효과를 그대로 사용할 수 있었다.

III. 레이어 채널 이미지 처리 연구

1. 레이어 채널의 기본 구성

일반적인 이미지의 RGBA가 32비트라는 점을 이용하여 구현한 기법이다. 일반적으로 그림 파일은 [그림 5]의 (a)와 같은 RGBA의 32비트로 총 4채널을 사용한다. 여기서 응용한 것은 32비트의 RGBA정보를 [그림 5]의 (b)처럼 16비트로 다운시켰다. 이렇게 되면 R은 4비트, B는 4비트, G는 4비트, A는 4비트 총 16비트가 된다. 그렇게 되면 16비트가 남게 되는데 남은 16비트는 R'는 4비트, G'는 4비트, B'는 4비트, L은 4비트로 레이어 채널을 생성하여 사용을 하였다.

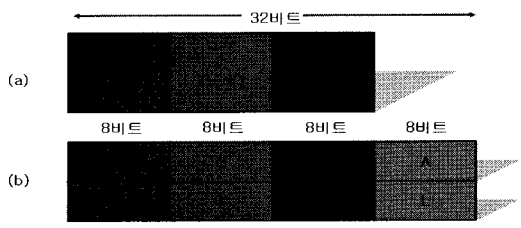


그림 5. 이미지 RGBA의 정보값 구조

여기서 L 값은 레이어 값으로 4비트가 된다. 각 비트는 다음을 의미한다.

- 첫 번째 비트는 플립핑(flipping. 그래픽에서 두 이미지를 순식간에 바꾸려는 행위)비트로 0이면 원본 이미지 데이터를 출력하고 1이면 레이어 이미지 데이터를 출력하게 하는 값이다.
- 두 번째 비트는 레이어 이미지의 원본 이미지 영역 보호 기능이다.
- 세 번째 비트는 레이어 이미지의 알파블랜딩 여부 기능 값이다.
- 네 번째 비트는 여유 비트값이다.

이렇게 하면 각각 8비트를 이루는 RGBA의 [그림 5]의 (b)처럼 상위 4비트는 원본 이미지 데이터가 되고 하위 4비트 레이어 채널이 된다. 아래는 레이어 채널 이미지의 데이터를 어떻게 시스템 메모리 혹은 비디오 메모리로 옮겨지는 지를 설명하는 알고리즘이다.

```
최종 출력될 이미지의 변수는 16비트의 resultImage라는 변수를 선언
실제 가지고 있는 레이어 채널 이미지의 변수형은
32비트 데이터형으로 layerChnnalImage라는 선언한다.
//먼저 이미지 데이터의 개수만큼 처리를 한다.
for( pointCount=0 ; pointCount<imageSize; pointCount ++)
//최종 resultImage 변수가 얻게 되는 이미지 정보는
//L채널의 첫 번째 비트값이 1이면 상위 4비트값을 얻게 되고
//0이 되면 하위 4비트 값을 갖게 된다.
resultImage[pointCount]  = layerChnnalImage[pointCount] >> 4*L(1);
```

일반적인 이것이 원본 이미지와 레이어 이미지를 구분하여 출력하는 기본 알고리즘이고 [그림 6]은 그 알고리즘 구조를 플로우 차트화한 것이다.

이렇게 하면 레이어의 색의 질은 32비트에서 16비트로 떨어지지만 이미지의 가로의 크기와 세로의 크기가 변하지 않고 원래 이미지의 형태를 유지할 수 있다.

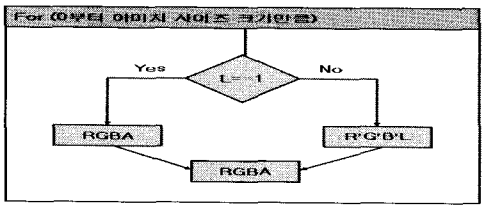


그림 6. 레이어 채널 이미지의 출력 기본 알고리즘

2. 팔레트를 이용한 레이어 채널의 구성

위의 문제점은 그래픽의 질이 32비트 칼라에서 16비트 칼라로 떨어진다는 점이다. 이것을 해결하기 위해서 8비트 칼라 포맷에서 사용하는 기능인 팔레트 기법을 사용하였다. 레이어 채널 이미지의 기본 구성은 순차적으로 데이터를 읽어오면서 그 중에 원본 데이터를 옮길지 아니면 레이어 이미지를 옮길지 결정해서 최종 시스템이나 비디오 메모리에 옮기는 것이 핵심이다. 이때 비트천이(2진수 데이터인 비트를 왼쪽이나 오른쪽으로 자리이동 시키는 수식 기법)를 이용하여 32비트 데이터를 16비트로 다운시키고 최종 결과 이미지도 16비트 이미지로 출력하였다. 이 전달 과정에서 팔레트를 이용하면 32비트 중 각각 상위 16비트 하위 16비트값이 결국엔 32비트 정보로 다시 환원되는 것이므로 [그림 7]에서 보는 것처럼 각각 상위 비트값과 하위 비트값이 지정하는 위치의 팔레트 데이터 안의 색을 불러와서 원래 32비트의 색상 값으로 시스템이나 비디오 메모리로 차례 차례 옮겨주었다.

그림 7. 팔레트를 이용한 레이어 채널 이미지 처리

```
for( pointCount=0 ; pointCount<imageSize; pointCount ++)
  resultImage[pointCount]  = layerChnnalImage[pointCount] >> 4*L(1);
```

위의 레이어 채널 이미지 기본 소스에서 밑줄 친 부분에서 resultImage는 32비트의 데이터 형으로 선언하고 아래와 같이 다시 구현하였다.

```
resultImage[pointCount]  = palletteIndex[layerChnnalImage[pointCount] >> 4*L(1)];
```

palletteIndex는 32비트의 색 중에서 65536개의 색을 가지고 있는 배열이다. 이렇게 되면 32비트 칼라 이미지를 그대로 사용하게 된다.

IV. 레이어 채널 이미지 출력 기법 개발

1. 레이어 채널 이미지의 기능

3D 영상 그래픽에서 이미지를 이용한 여러 가지 특수 효과를 사용하기도 한다. 그 특수 효과들은 두 장 이상의 이미지를 서로 계산해서 최종 이미지를 얻어내는데, 많은 계산량 때문에 속도 저하의 원인이 되기도 한다. 따라서 레이어 채널 이미지의 기능을 이용하여 같은 특수 효과를 적은 계산으로 구현하였다.

본 연구에서는 레이어 채널 이미지가 표현 할 수 있는 이미지 특수 효과와 일반적으로 구현되고 있는 특수 효과와 비교하여 구현하였다.

2. 플립핑 이미지 기능

이미지의 일부를 변경시키는 것으로 일반적으로 이미지를 플립핑하는 방법은 다음과 같이 표현한다. 바탕이 될 기본 이미지를 destImage라고 정의하고 이미지의 일부가 변경될 이미지를 sourImage라고 정의한다면 아래와 같다.

```
for( pointCount=0 ; pointCount<imageSize; pointCount ++)
        resultImage[pointCount]  = destImage[pointCount];

pointX'=0;
pointY'=0;
for( pointY=y ; pointY<y'; pointY ++)
{
        pointY'++
        for( pointX=x ; pointX<x'; pointX ++)
        {
                pointX'++
                if(sourImage[pointX'][pointY']!=colorKey)
                resultImage[pointX][pointY]  = sourImage[pointX'][pointY'];
        }
}
```

이렇게 하면 [그림 8]의 (a)를 보면 알 수 있듯이 바탕이 될 이미지 데이터를 전부 전송한 다음에 다른 이미지의 플립핑을 원하는 데이터의 원하는 부분을 영역으로 잡고 표현하고자 하는 색상을 원본 이미지의 위치에 덮어씌운다. 일반적인 이미지를 사용했을 경우는 위와 같고 레이어 채널 이미지를 사용하여 플립핑 이미지를 구현하기 위해선 원본 이미지는 상위 비트에 저장시키고 바뀔 이미지는 레이어 이미지로 저장을 한다. 그리고 바뀔 정보가 있는 위치는 L채널 첫 번째 비트에 세팅을 한다. 이와 같이 위의 기본 식으로 기존의 복잡한 방법을 레이어 채널 이미지를 이용하면 [그림 8]에 (b)에서 볼 수 있는 것처럼 한 번의 루프로 표현 할 수 있다.

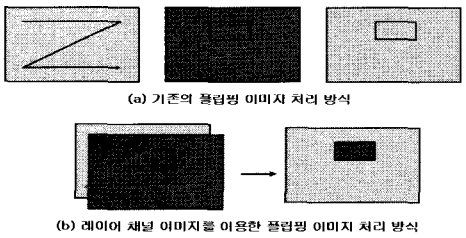


그림 8. 이미지의 플립핑 처리

이 기법을 응용하면 라이트맵이나 멀티텍스쳐 그리고 캐릭터나 이미지의 외각선을 쉽게 구현 할 수 있었다.

3. 레이어 이미지의 변위 기능

레이어 채널 이미지의 L(2)값을 이용하여 레이어 이미지의 출력 위치를 변화시킬 수 있다. 일반적으로 두 장의 이미지를 변위 차이가 나게 겹치는 방법은 아래와 같다. 먼저 밑에 겹쳐질 이미지 데이터를 전송 받았다.

```
for( pointCount=0 ; pointCount<imageSize; pointCount ++)
  resultImage[pointCount]  = destImage[pointCount];
```

먼저 그려질 데이터가 입력된 resultImage 데이터에 겹칠 이미지를 위치를 변경한 다음에 덮어 그린다. 이때 칼라킷값(Color Key. 직사각형으로 된 이미지 중에서 필요 없는 부분을 제외 시켜주기 위해서 지정된 색)은 데이터는 덮지 않고 다음 데이터로 넘어간다.

```
pointX'=0;
pointY'=0;
for( pointY=0 ; pointY<imageHeight; pointY ++)
{
        pointY' = pointY +deltaY;
        for( pointX=0 ; pointX<imageWidth; pointX ++)
        {
                pointX' = pointX +deltaX
                if(sourImage[pointX][pointY]!=colorKey)
                resultImage[pointX'][pointY'] = sourImage[pointX][pointY];
        }
}
```

위의 같은 것을 레이어 채널 이미지에서 레이어 이미지의 위치를 변화 시켜서 사용하는 것은 아래의 식과 같이 구현했다.

```
for( pointCount=0 ; pointCount<imageSize; pointCount ++)
{
        switch( L[2] )
        {
        case 0:
                deltaPoint= pointCount + (deltX + (imageWidth * deltY) * L[2];
                resultImage[pointCount] = layerChnnalImage[deltaPoint];
                break;
        case 1:
                deltaPoint= pointCount;
                resultImage[pointCount] = layerChnnalImage[deltaPoint] >> 4;
                break;
        }
}
```

L의 두 번째 비트 값은 원본 이미지 데이터와 레이어 이미지를 같이 표현할 것인지, 아니면 원본 이미지만 표현 할 것인지를 판단해 주는 값이다. 즉 L(2)값이 0이면 현재 위치에 위치가 변한 레이어 이미지의 데이터 값을 얻고 그렇지 않으면 위치가 변하지 않은 원본 데이터만 표현했다.

V. 구현 및 결과

본 연구에서는 3D 영상 안에서 표현되고 있는 특수 효과를 위한 디지털영상 제작용 이미지 파일인 레이어 채널 이미지 처리 소프트웨어를 구현하였다.

[그림 9]는 이러한 시스템을 보여준다.

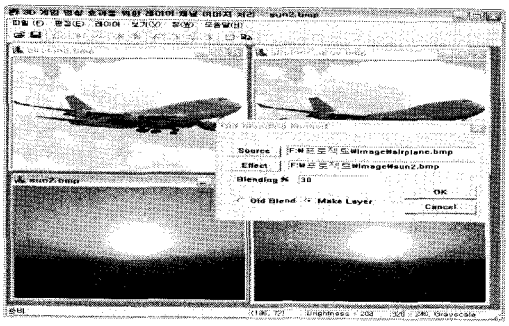


그림 9. 레이어 채널 이미지 처리 소프트웨어

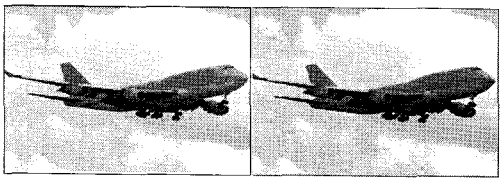


그림 10. 입력 영상

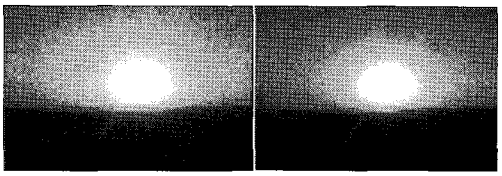


그림 11. 블랜딩 영상

[그림 10]은 이번 실험에서 사용한 영상이고 [그림 11]은 블랜딩 효과를 주기 위한 영상이다.

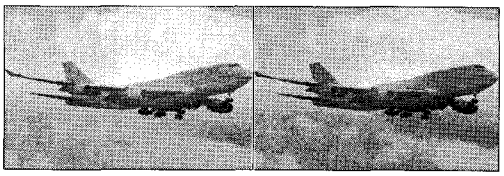


그림 12. 결과 영상

[그림 10]은 원 영상으로 특수 처리되기 전의 영상이다. [그림 11]의 이미지는 블랜딩 효과를 나타내기 위한 효과 영상이다. 이러한 효과영상을 원영상에 30%의 블랜딩 효과를 주면 [그림 12]와 같은 결과가 나타내게 된다. 위의 영상의 기존의 이미지 파일의 기본 구조가 아닌 새로운 레이어 채널 이미지 파일 구조이다. 그리고 이미지가 레이어 채널에 잘 적재 되었는지 확인하기 위해서 레이어 채널 이미지를 볼 수 있는 기능을 만들어 넣었다. [그림 13]은 레이어 채널에 있는 이미지를 보여준다. [그림 13]은 [그림 12]의 파일에 있는 레이어 이미지 이다.

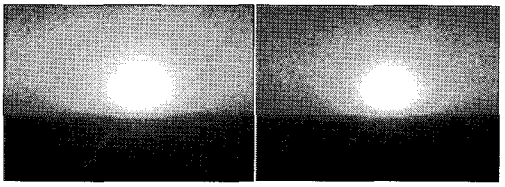


그림 13. 레이어 이미지

시스템의 처리의 결과 [그림 12]의 영상파일과 [그림 14]의 영상파일의 구조는 다르다. 레이어 채널에 이미지 한 장의 이미지가 더 적재되어 있기 때문에 이미지

용량 크기도 다르다.

표. 영상 이미지의 용량 크기

구 분	흑백 영상	컬러 영상
기본 이미지	77,880B	230,456B
레이어 이미지	77,880B	230,456B
레이어 채널이미지	154,678B	460,854B

흑백 영상와 컬러 영상의 용량크기에서 각각 기본 이미지는 77,880B / 230,456B 이고 레이어 이미지는 77,880B / 230,456B 이지만 레이어 채널 이미지는 154,678B / 460,854B 이다. 그리고 흑백 영상의 경우 기본 이미지와 레이어 이미지 용량의 합은 155,760B에 해당하고 레이어 채널 이미지의 경우 154,678B 이다. 또한 컬러 영상의 경우도 기본 이미지와 레이어 이미지의 용량이 460,912B 이지만 레이어 채널 이미지의 경우 460,854B 로 각각 1,082B / 58B 로 감소되었다. 이 이미지는 한 장에 해당하는 것이다. 3D 영상을 보여주기 위해 수천에서 수만의 영상들이 처리된다. 따라서 레이어 채널 이미지를 사용함으로서 3D 영상에 사용되는 영상의 크기를 상당부분은 줄일 수 있다. 마지막으로 플립핑을 개선한 예도 보여주고자 한다. 아래의 [그림 14]에서 보듯이 레이어 영상에 임의의 아바타를 입력할 수 있다. 그러나 플립핑할 영역이 미리 지정된 레이어를 사용함으로써 기존의 개별적인 영상 처리를 조금 더 향상시킬 수 있다.

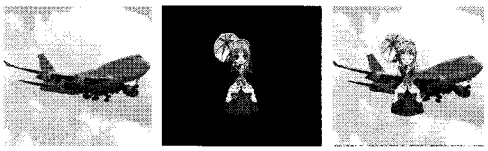


그림 14. 기본 영상에 중간 아바파를 플립핑한 영상

Ⅵ. 결 론

3D 영상 효과를 위한 레이어 채널 이미지 처리 기법에 관한 연구는 이미지를 이용한 특수효과에 많은 이미지 데이터를 사용하고, 처리 속도의 저하를 가져다주는 기존의 방식에 이미지 자체가 특수 효과를 처리 할 수 있는 정보로 재구성하는 방법을 제공함으로써 새로운 기법을 만들어 사용할 수 있도록 수행하고, 지금까지 소개된 기존의 이미지는 단지 이미지 자체가 특별한 역할을 하는 것은 아니고 단지 데이터로서의 기능만을 제공하였으며, 이러한 이미지를 기능성 있는 이미지 데이터로 가공하여 처리하여 3D 영상 효과를 위한 레이어 채널을 구성함으로서 더 많은 데이터 처리나 처리 속도 저하를 줄일 수 있음을 실험을 통해 확인할 수 있었다.

*** 개발 결과**

- 영상포맷을 변경하여 레이어 채널을 숨기는 기술.
- 일반 영상뷰어에서도 변경된 포맷의 3D 영상을 볼 수 있도록 개발.
- 레이어 채널과 영상을 묶음으로써 재사용성 높임.
- 모든 3D 영상 프로그램에 이용 가능하도록 만듦.

*** 개발 효과**

- 영상 로드시에 영상과 레이어 채널을 동시에 불러드림으로서 처리 속도 향상시킴.
- 3D 영상에 레이어 채널을 삽입함으로써 레이어 채널 영상을 위한 소스 저장 공간을 줄임.
- 3D 영상과 레이어 채널의 영상을 한 번에 다룰 수 있게 되어 효과적인 이펙트 표현 가능.
- 실제 애플리케이션이 될 수 있는 멀티미디어 영상 등에 효과적으로 이용이 가능.

*** 개발(실험) 과정**

본 관련 그림들은 실제 원 영상과 레이어 영상을 어떻게 처리하는 지 보여주는 차원에서 개발된 프로그램 사진들이다. 크게 네가지 과정으로 보여줄 수 있는데, 첫째는 레이어가 삽입된 영상을 만드는 그림[그림 15] [그림 16] 레이어가 삽입된 영상을 처리하는 뷰어[그림 17]~[그림 20], 레이어가 삽입된 영상의 블렌딩 처리장면[그림 21]~[그림 23]이다.

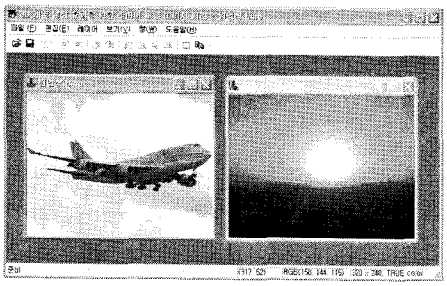


그림 15. 원영상과 레이어로 쓸 영상을 불러 온 장면

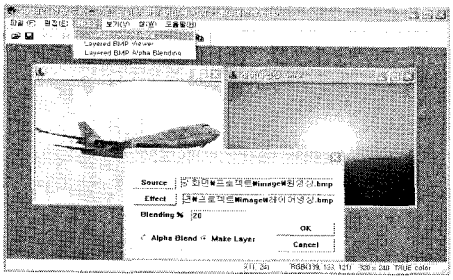


그림 16. 두 영상을 가지고 레이어가 삽입된 비트맵 영상을 만드는 과정

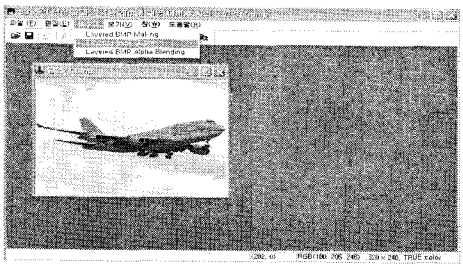


그림 17. 레이어 삽입 비트맵 영상 뷰어 메뉴

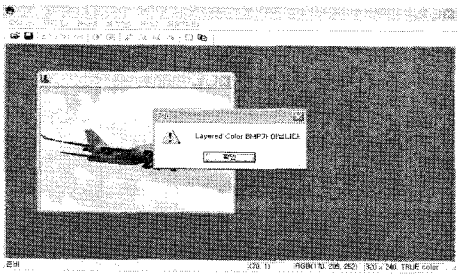


그림 18. [그림 16]의 결과로서 레이어가 삽입된 영상이 아님을 출력

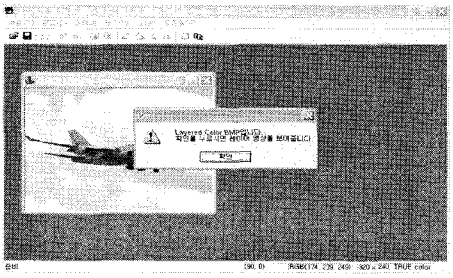


그림 19. [그림 17]에서 만든 영상을 레이어 비트맵 뷰어 메뉴로 볼 경우 확인된 장면

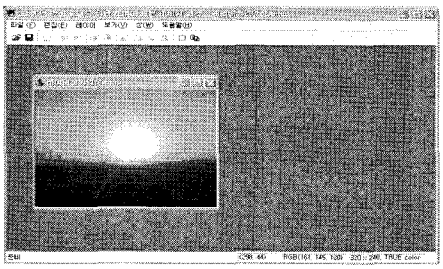


그림 20. [그림 19]에서 확인을 누를 경우 숨겨진 레이어 영상을 보여줌

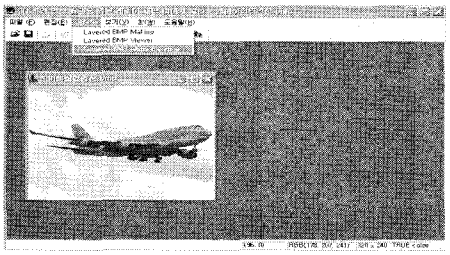


그림 21. 레이어가 삽입된 영상에 대하여 알파 블렌딩을 적용하는 메뉴

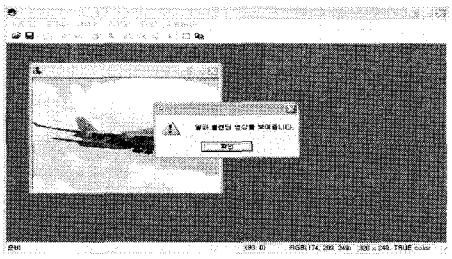


그림 22. 레이어 삽입 영상이 맞는 경우 그림과 같은 대화 상자가 뜸

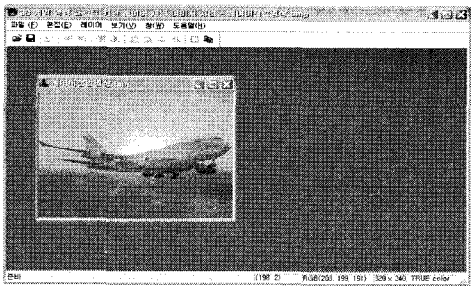


그림 23. [그림 22]에서 확인을 누를 경우 원영상과 숨겨진 레이어의 혼합 영상이 나옴

* 적용 가능한 영화영상 사례들

3D 영상중 실시간 영화의 한 장면처럼 현실감 있게 빛이나 조명 처리 표현의 어려운 부분을 이런 비슷한 효과들을 이용하여 표현 가능한 사례들이다[그림 24]~[그림 26].

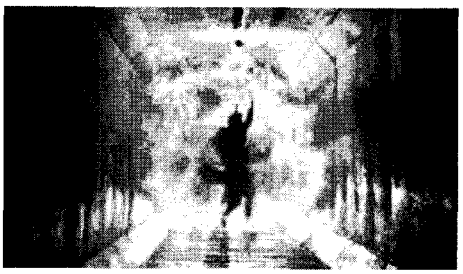


그림 24. 영화의 폭파 장면을 표현한 사례

[그림 24]처럼 폭파의 강렬한 빛효과를 표현할 수 있다.

그림 25. 영화의 눈부심 장면을 표현한 사례

[그림 25]처럼 빛의 눈부심 시각효과를 표현할 수 있다.

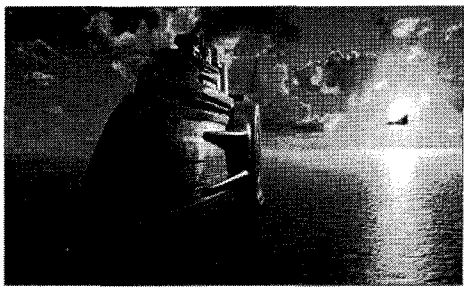


그림 26. 영화의 노을빛 장면을 표현한 사례

[그림 26]처럼 붉은 노을빛 효과를 표현할 수 있다. 3D 영상에서 표현하기 힘들었던 연출 등을 이제 동영상에서 뿐만아니라 이런 기법들을 영화에서도 이용하면 실시간으로 처리할 수 있어 3D 영상을 즐길 수 있을 것이다.

참 고 문 헌

[1] T. S. Roger, *Computer Graphics Dictipnary* CHARLES RIVER MEDIA, 2001.

[2] S. R. Richard and J. S. Michael, *OpenGL Super Bible Second Edition*, 인포북, 2001.

[3] W. Mason, N. Jackie, D. Tom, and S. Dave, *OpenGL Programming Guide Third Edition*, ADDISON-WESLEY, 2001.

[4] P. Adrain and R. Dan, *Advanced 3D Game Programmin using DirectX*, 민프레스, 2001.

[5] Crooks, Clayton E, *3D Game Programming With DirectX 8.0*, Charles River Media, 2001.

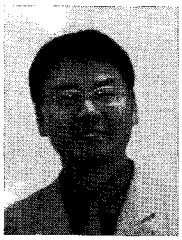
[6] 메이슨, 남기혁, *OpenGL 프로그래밍 가이드제3판*, 인포북, 2003.

[7] 김용준, *3D 게임 프로그래밍 IT EXPERT*, 한빛미디어, 2003.

[8] 최학현, 이명학, *게임 프로그래밍*, 진한M&B, 2006.

저 자 소 개

최 학 현(Hak-Hyun Choi) 정회원



- 2005년 : 고려대학교 영상전공 (박사수료)
- 2007년 : 서울대학교 문화콘텐츠(GLA)
- 2007년 : 문화콘텐츠연구센터 (책임교수)

<관심분야> : 디지털콘텐츠(Digital Contents)

김 정 희(Jung-Hee Kim) 정회원

- 2001년 : 숙명여자대학교(학사)
- 2007년 : 성균관대학교(박사과정)
- 2007년 : 문화콘텐츠연구센터 (객원연구원)

<관심분야> : 문화콘텐츠

이 명 학(Myung-Hak Lee) 정회원

- 2005년 : 숭실대학교(석사)
- 2007년 : 숭실대학교(박사과정)
- 2007년 : 문화콘텐츠연구센터 (객원연구원)

<관심분야> : 미디어