(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.
G06T 1/00 (2006.01)
G06T 7/00 (2006.01)

(11) 공개번호 10-2006-0101223
(43) 공개일자 2006년09월22일

(21) 출원번호 10-2006-0008912
(22) 출원일자 2006년01월27일

(30) 우선권주장 11/083,378 2005년03월18일 미국(US)

(71) 출원인 마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이

(72) 발명자 왕, 총-지
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
버치, 워렌 엘.
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

(74) 대리인 주성민
백만기
이중희

심사청구 : 없음

(54) 3D 모델을 사용하여 2D 전이를 발생하는 방법 및 시스템

요약

제1 미디어 객체와 제2 미디어 객체 간에 2D 전이를 발생하는 시스템 및 방법이 제공된다. 원하는 2D 전이 패턴에 대응하는 3D 모델이 정의된다. 그래픽 처리 유닛(GPU) 또는 일련의 명령어는 정의된 3D 모델의 함수로서 제1 미디어 객체와 제2 미디어 객체 간의 전이를 발생한다. 전이 발생기는 제1 미디어 객체의 복수의 정점을 제1 메쉬에 매핑하여 제1 매핑된 미디어 객체를 생성하며, 여기서 제1 메쉬는 3D 모델과 관련이 있다. 전이 발생기는 제2 미디어 객체의 복수의 정점을 전이 패턴 메쉬에 매핑하여 제2 매핑된 미디어 객체를 생성하며, 여기서 전이 패턴 메쉬는 3D 모델과 관련이 있다. GPU 또는 일련의 명령어는 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체로부터 전이를 렌더링한다.

대표도

도 1

색인어

그래픽 애니메이션, 2D 전이, 미디어 객체, 렌더링, 패턴 메쉬

명세서

도면의 간단한 설명

도 1은 본 발명의 일 실시예에 따라 제1 미디어 객체와 제2 미디어 객체 간의 2차원 전이를 발생하는 시스템을 나타낸 블록도.

도 2a는 본 발명의 일 실시예에 따른 3차원 모델을 나타낸 도면.

도 2b는 본 발명의 일 실시예에 따른 직교 투영 변환(orthogonal projection transform)을 나타낸 도면.

도 2c는 본 발명의 일 실시예에 따른 서로 다른 차원에서의 전이 패턴의 뷰를 나타낸 도면.

도 3은 본 발명의 일 실시예에 따른 전이 패턴 메쉬의 구현을 나타낸 도면.

도 4a 내지 도 4c는 본 발명의 일 실시예에 따라 3차원 모델을 사용하는 서클 와이프 전이 패턴(circle wipe transition pattern)의 구현을 나타낸 도면.

도 5a 및 도 5b는 본 발명의 일 실시예에 따른 예시적인 전이 패턴의 구현을 나타낸 도면.

도 6은 본 발명의 실시예에 따라 샤프 전이 패턴 엣지를 갖는 제1 미디어 객체와 제2 미디어 객체 간의 2차원 전이를 발생하는 방법을 나타낸 플로우차트.

도 7은 본 발명의 일 실시예에 따라 제1 미디어 객체와 제2 미디어 객체 간의 전이에 대한 부가의 효과를 생성하는 방법을 나타낸 플로우차트.

도 8은 본 발명이 구현될 수 있는 적당한 컴퓨팅 시스템 환경의 일례를 나타낸 블록도.

도 9 및 도 9a는 본 발명의 일 실시예에 따른 알파 맵의 예를 나타낸 도면.

<도면의 주요 부분에 대한 부호의 설명>

102, 122: 제1 미디어 객체

104, 124: 제2 미디어 객체

106: 제1 메쉬

108: 인터페이스

110: 전이 패턴 메쉬

112: 전이 발생기

114: 제1 프로세서

116: 프레임 버퍼

118: 제2 프로세서

120: 디스플레이

126: 전이

128: 알파 맵

130: 제3 미디어 객체

132: 디졸브 맵

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명의 실시예들은 일반적으로 그래픽 애니메이션 및 처리 분야에 관한 것이다. 상세하게는, 본 발명의 실시예들은 3차원 모델을 사용하여 2차원 전이를 발생하는 것 및 그래픽 처리 유닛의 처리 기능들에 관한 것이다.

일반적인 그래픽 처리에서, 소프트웨어 애플리케이션은 비트맵 텍스트(bitmapped text)(예를 들어, 트루타입 폰트(TrueType font)), 이미지 및 다른 그래픽 요소를 디스플레이 또는 인쇄하기 위해 하드웨어 인터페이스(예를 들어, 그래픽 장치 인터페이스(GDI))를 사용한다. 예를 들어, GDI는 적절한 디스플레이 드라이버를 호출하고 이 드라이버에 드로잉될 아이템에 관한 정보를 전달함으로써 화면 상에 일관된 방식으로 아이템(대화 상자, 버튼, 및 기타 그래픽 요소 등)을 드로잉하는 일을 맡고 있다. GDI를 이용하기 위해, 소프트웨어 프로그램은 일반적으로 GDI 함수를 호출하도록 이들 소프트웨어 애플리케이션을 구성하기 위해 소프트웨어 개발 킷(SDK) 라이브러리에 제공된 루틴 등의 일련의 루틴을 사용한다. 예를 들어, SDK 라이브러리는 아이콘 연산, 직사각형 연산, 기타 등등의 함수들을 포함한다.

현재의 컴퓨터 시스템 및 구성 하에서, 대부분의 GDI 특징은 중앙 처리 장치(CPU)에 의해 실행된다. 그 결과, 다른 프로세스를 실행하는 것 외에도, CPU는 2D 이미지, 비디오 스트림 또는 기타 등등을 렌더링하는 것 등 그래픽 요소 및 이미지를 처리해야만 한다. 부가의 그래픽 처리는 CPU 및 컴퓨터 시스템의 전체 성능에 떨어뜨리는데 그 이유는 그래픽 처리가 픽셀 또는 텍스처 계산, 이미지의 렌더링 또는 애니메이션, 또는 기타 등등의 엄청난 그래픽 파라미터 계산을 필요로 하기 때문이다. 이들 그래픽 또는 이미지에 대한 CPU 처리가 적절하지만, 그래픽 또는 이미지 처리의 개선이 바람직하다.

게다가, 대부분의 현재 시스템 또는 애플리케이션은 그래픽 효과를 처리하는 데 2차원 알고리즘을 사용한다. 예를 들어, 사용자는 2개의 미디어 객체를 디스플레이하는 결과 시퀀스가 매끄러운 전이를 갖도록 2개의 미디어 객체 또는 이미지 간의 전이를 삽입할 수 있다. 기존의 시스템에서, CPU는 결과 시퀀스를 디스플레이하는 것 이외에도 알고리즘 및 프로그래밍 명령어를 실행하는 데 자원 및 시간을 필요로 한다.

따라서, 2차원 그래픽 효과를 생성하는 데 GPU의 3D 모델 및 고정 파이프라인(fixed pipeline) 함수를 사용함으로써 그래픽 처리 작업을 그래픽 처리 유닛(GPU)에 위임 또는 넘기는 시스템 및 방법이 바람직하다. 다른 대안으로서, 3차원 좌표계에서 2차원 효과를 처리하기 위한 일련의 명령어를 사용하여 2차원 그래픽 효과를 생성하는 시스템 및 방법이 요망된다.

발명이 이루고자 하는 기술적 과제

본 발명의 실시예들은 3차원 좌표계를 사용하여 2차원 전이를 구현함으로써 종래 기술의 하나 이상의 단점을 극복한다. 행렬 계산, 픽셀 또는 텍스처 계산을 수행하기 위해 2차원 전이에 관한 변환된 데이터를 CPU로 전송하지 않고, 본 발명의 실시예들은 GPU가 데이터를 보다 효율적으로 처리할 수 있도록 데이터를 CPU로 전송하지 않고 데이터를 GPU로 전송한다. GPU는 디스플레이할 준비를 하기 위해 미디어 객체, 애니메이션 또는 다른 그래픽 요소를 비디오 버퍼 상으로 렌더링할 수 있다. 다른 대안으로서, 일련의 컴퓨터 실행가능 명령어 또는 소프트웨어 애플리케이션은 2차원 효과를 생성하기 위해 3D 좌표계에서 데이터를 처리할 수 있다. 따라서, 본 발명의 실시예들은 버스 트래픽, CPU 부하를 감소시키고, 전이, 애니메이션 또는 다른 그래픽 변환의 처리의 렌더링 시간을 단축시키는 데 도움을 준다.

게다가, 본 발명의 실시예들은 미디어 객체 또는 이미지들 간의 어떤 전이가 CPU 시간을 덜 차지하게 만들고 또 새로운 시각적 변형(visual variant) 및 전이 패턴을 개발하는 것을 보다 효율적이도록 만든다. 이 디자인은 와이프 패턴(wipe pattern), 렌더링 기술 및 파라미터를 혼합함으로써 사용자에게 풍부한 시각적 변형의 컬렉션을 제공한다. 본 발명의 실시예들은 또한 현재 시판 중인 대부분의 기계에서 2차원 그래픽 효과를 생성하는 동안 GPU 고정 함수 파이프라인(fixed

function pipeline)을 사용하여 3차원 미디어 객체를 실행 및 처리한다. 다른 대안으로서, 본 발명의 실시예들은 2차원 그래픽 효과를 생성하기 위해 3차원 미디어 객체를 처리하는 데 일련의 명령어 또는 소프트웨어 애플리케이션의 고정 함수 파이프라인을 사용할 수 있다.

본 발명의 한 측면에 따르면, 그래픽 처리 유닛을 사용하여 제1 미디어 객체와 제2 미디어 객체 간의 2차원 전이를 발생하는 방법이 제공된다. 원하는 2차원 전이 패턴에 대응하는 3차원 모델이 정의된다. 그래픽 처리 유닛을 사용하여 상기 정의된 3차원 모델의 함수로서 제1 미디어 객체와 제2 미디어 객체 간의 전이가 발생된다.

본 발명의 다른 측면에 따르면, 제1 미디어 객체와 제2 미디어 객체 간의 2차원 전이를 발생하는 시스템이 제공된다. 인터페이스는 제1 미디어 객체, 제2 미디어 객체, 및 전이 패턴 메쉬를 수신한다. 전이 발생기는 제1 매핑된 미디어 객체를 생성하기 위해 수신된 제1 미디어 객체의 복수의 정점을 제1 메쉬에 매핑한다. 이 전이 발생기는 제2 매핑된 미디어 객체를 생성하기 위해 수신된 제2 미디어 객체의 복수의 정점을 전이 패턴 메쉬에 매핑한다. 그래픽 프로세서는 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체로부터 제1 미디어 객체와 제2 미디어 객체 간의 전이를 렌더링한다.

본 발명의 또 다른 측면에 따르면, 제1 미디어 객체와 제2 미디어 객체 간의 전이를 발생하는 시스템이 제공된다. 제1 프로세서 또는 제1 일련의 명령어는 제1 미디어 객체 및 상기 제2 미디어 객체를 렌더링한다. 인터페이스는 제1 미디어 객체, 제2 미디어 객체 및 전이 패턴 메쉬를 수신한다. 전이 발생기는 제1 매핑된 미디어 객체를 생성하기 위해 수신된 제1 미디어 객체의 복수의 정점을 제1 메쉬에 매핑한다. 이 전이 발생기는 제2 매핑된 미디어 객체를 생성하기 위해 수신된 제2 미디어 객체의 복수의 정점을 수신된 전이 패턴 메쉬에 매핑한다. 제2 프로세서 또는 제2 일련의 명령어는, 제1 프로세서 또는 제1 일련의 명령어와 독립적으로, 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체로부터 제1 미디어 객체와 제2 미디어 객체 간의 전이를 렌더링한다.

다른 대안으로서, 본 발명은 여러가지 다른 방법 및 장치를 포함할 수 있다.

다른 특징들은 부분적으로는 명백하고 부분적으로는 이후에 지적될 것이다.

발명의 구성 및 작용

먼저 도 1을 참조하면, 블록도는 본 발명의 일 실시예에 따라 제1 미디어 객체(102)와 제2 미디어 객체(104) 간의 2차원(2D) 전이를 발생하는 시스템(100)을 나타낸 것이다. 전이(transition)란 패턴에 따라 제1 미디어 객체로부터 제2 미디어 객체로의 디스플레이 상에서의 점진적인 변화를 말한다. 예를 들어, 제1 미디어 객체(102)가 백색 이미지이고 제2 미디어 객체(104)가 흑색 이미지이며 사용자가 백색 이미지로부터 흑색 이미지로 전이하기를 원하는 것으로 가정하자. 전이의 한 구현은 흑색 이미지가 전체적으로 보이고 백색 이미지를 완전히 대체할 때까지 좌측에서 우측으로 흑색 이미지가 점차적으로 백색 이미지를 대체하는 것인 좌측에서 우측으로의 와이프 전이(wipe transition)일 수 있다. 이러한 좌측에서 우측으로의 와이프 전이는 수직 라인이 백색 이미지를 통해 좌측에서 우측으로 수평으로 이동하고 흑색 이미지가 백색 이미지를 완전히 대체할 때까지 수직 라인이 좌측에서 우측으로 점차적으로 이동할 때 흑색 이미지를 수직 라인의 좌측에 나타나게 하는 것 등의 패턴을 포함할 수 있다. 수평 라인이 수직으로 이동하는 것, 2개의 수직 라인이 중앙으로부터 이동하는 것, 또는 기타 등등의 다른 패턴이 구현될 수 있다.

제1 프로세서(114), 제2 프로세서(118), 디스플레이(120), 및 저장 장치(도시 생략)를 포함하는 시스템(100)은 도 8에 나타낸 것과 같은 시스템(130) 또는 다른 컴퓨터화된 시스템일 수 있다. 다른 실시예에서, 시스템(100)은 3차원(3D) 객체를 처리 및 디스플레이하기 위한 애플리케이션 프로그램 인터페이스(API)를 포함한다. 예를 들어, 이러한 API는 Microsoft Corporation의 Direct3D®, Apple Computer, Inc.의 QuickDraw®, 또는 Silicon Graphics Incorporated의 OpenGL®일 수 있다. 제1 프로세서(114)는 처리를 위한 CPU, 다른 중앙 또는 메인 프로세서, 제1 일련의 컴퓨터 실행가능 명령어, 또는 소프트웨어 애플리케이션일 수 있다. 제2 프로세서(118)는 3D 모델에 관한 데이터를 처리하기 위한 GPU, 다른 2차 프로세서, 제2 일련의 컴퓨터 실행가능 명령어 또는 소프트웨어 애플리케이션일 수 있다.

처음에, 제1 미디어 객체(102) 및 제2 미디어 객체(104)는 디스플레이(120) 상에 디스플레이된다. 예를 들어, 제1 미디어 객체(102) 또는 제2 미디어 객체(104)는 디스플레이(120) 상에 디스플레이될 정지 영상, 비디오 스트림 또는 다른 그래픽 요소(예를 들어, 아이콘, 버튼, 웹 페이지, 기타 등등)에 대한 파일 또는 파일들의 컬렉션일 수 있다. 사용자는 타임 라인에서 제1 미디어 객체(102)와 제2 미디어 객체(104) 간의 매끄러운 또는 다른 흥미로운 전이를 생성하고자 할 수 있다. 흥미로운 전이를 생성하는 것 이외에, 사용자는 보는 사람이 전이 및/또는 효과가 아니라 미디어 객체를 통해 표현된 이야기에 집중해주기를 원할 수 있다. 그 자체로서, 사용자는 미묘한 효과 및 변화를 갖는 전이를 생성하고자 할 수 있다. 예를 들어,

사용자가 트로피를 증정하는 등 보는 사람에게 흥분을 가져다주거나 영광을 같이 하기를 원하는 것으로 가정하면, 사용자는 트로피 증정이 눈에 잘 띄도록 제1 미디어 객체(102)와 제2 미디어 객체(104) 간의 전이를 생성함으로써 트로피가 장면에 장엄하게 등장하도록 하고 싶어할 수 있다. 반면에, 사용자는 보는 사람을 영화 장면의 끝으로 전이시키기 위해 미묘한 전이(소프트 엣지(soft edge), 디졸브 엣지(dissolve edge), 및/또는 팬시 엣지(fancy edge) 전이 등)를 발생할 수 있다.

제1 미디어 객체(102) 및 제2 미디어 객체(104)는 인터페이스(108)에 의해 수신된다. 예를 들어, 인터페이스(108)는 제1 미디어 객체(102) 및 제2 미디어 객체(104)를 수신하는 소프트웨어 애플리케이션의 그래픽 사용자 인터페이스(GUI) 또는 다른 인터페이스 수단일 수 있다. 제1 미디어 객체(102) 및 제2 미디어 객체(104)는 통신 네트워크(예를 들어, 시스템 버스, 인터넷, 인트라넷, 기타 등등) 등의 직접 전송(direct transmission)을 통해 비휘발성 저장 장치, 휘발성 저장 장치 기타 등등으로부터 인터페이스(108)로 입력될 수 있다.

한 예에서, 사용자는 제1 미디어 객체(102)와 제2 미디어 객체(104) 간에 와이프 전이를 생성할 수 있다. 와이프 전이가 2D 그래픽 효과이지만(예를 들어, 제2 미디어 객체(104)가 제1 미디어 객체(102)를 대체하기 위해 컴퓨터 디스플레이를 가로질러 좌측에서 우측으로 이동함), 본 발명의 실시예들은 임의의 주어진 전이에 대한 3D 모델(예를 들어, x, y, z 좌표계)을 정의한다. 예를 들어, 도 2a는 본 발명의 일 실시예에 따른 3D 모델(200)을 나타낸 블록도이다. 모델(200)은 3개의 축(x축, y축 및 z축)을 갖는 3D 뷰 공간(view space)(204)을 보여준다. 다른 실시예에서, 3D 뷰 공간(204)은 디스플레이(120)의 여러가지 화면 종횡비(예를 들어, 4x3, 16x9)에 적합하도록 구성되어 있다. 모델(200)은 또한 보는 사람이 z축과 반대 방향(화살표(212)로 나타냄)으로 보고 있음을 나타내기 위해 눈(202)을 나타내고 있다. 환언하면, z축은 보는 사람쪽을 향하고 있다.

이제 도 2b를 참조하면, 도면은 본 발명의 일 실시예에 따라 2D 미디어 객체를 3D 모델에 매핑하기 위한 직교 투영 변환(214)을 나타낸 것이다. 변환(214)은 수평 엣지를 따라 있는 x축 및 x축에 수직인 y축을 갖는 2D 정사각형이다. 일 실시예에서, 변환(214)의 길이 및 폭은 가시 범위가 0 내지 1이 되도록 1 단위로 설정되어 있다. 즉, 변환(214)은 1x1 평방 단위이다. 2D 형상을 3D 모델에 매핑하기 위해 다른 형상 또는 다각형(예를 들어, 삼각형, 직사각형 기타 등등)이 사용될 수 있음을 잘 알 것이다. z축은 x-y 평면 상에 위치하고 있는 눈(216)쪽으로 향하고 있다. 도 2b에 도시한 예처럼, 눈은 위치(0.5, 0.5, 1.0)에 있다(예를 들어, x-y 평면의 상측으로부터 봄). 환언하면, 2D 미디어 객체가 3D 모델에서의 3D 좌표계와 관련한 데이터로 표현될 수 있도록, 2D 미디어 객체는 변환(214)(즉, 0,0 -> 1.0, 1.0 공간)에 매핑될 수 있다. 이러한 데이터는 2D 그래픽 효과를 생성하기 위해 GPU, 일련의 컴퓨터 실행가능 명령어, 또는 소프트웨어 애플리케이션에 의해 처리될 수 있다.

도 2c는 본 발명의 일 실시예에 따라 여러가지 차원에서의 전이 패턴의 뷰를 나타낸 것이다. 예로서, 뷰(206, 208, 210)는 서로 다른 뷰 공간에서의 휠 전이(wheel transition)를 나타낸 것이다. 측면 투시도(206)는 x-y-z 좌표 기호로 나타낸 바와 같이, 3D 모델에서의 휠 전이(wheel transition)이다. 3D 변환을 사용하는 많은 기존의 시스템에서, 투시도(206)는 사람의 눈에 가까운 객체를 사람의 눈으로부터 멀리 있는 객체보다 크게 함으로써 사람의 시각을 시뮬레이션한다. 반면에, 정투상도(orthogonal view)(208)는 x-y 좌표 기호로 표시한 바와 같이, 2D에서의 휠 전이를 조감도에서의 타원으로 평평하게 만든다. 이것은 2D 전이로서 제1 미디어 객체(102)로부터 제2 미디어 객체(104)로 전이할 때 디스플레이되는 것처럼 휠 전이의 예시적인 뷰이다. 반면에, z축으로부터의 전이 패턴 메쉬 뷰(210)는 전이 패턴 메쉬(110)가 적용된 후의 휠 전이의 뷰이다.

전이 패턴 메쉬 뷰(210)에 대해 더 설명하기 위해, 도 3은 본 발명의 일 실시예에 따라 제1 미디어 객체(304)의 중앙으로부터 제1 미디어 객체(304)를 제2 미디어 객체(306)로 대체시키는 와이프 전이 등의 전이 패턴 메쉬(110)의 구현을 나타낸 도면이다. 예를 들어, 도 3에 도시된 바와 같이, 제1 미디어 객체(304)는 라인으로 채워진 이미지이고 제2 미디어 객체(306)는 백색 이미지이다. 눈(302)으로 나타낸 방향으로부터 보고 있는 사용자는 제2 미디어 객체(306)를 제1 미디어 객체(304)의 중앙으로부터 나타나도록 하고 싶다. 처음에, 사용자는 도 3에 참조 번호(308)로 나타낸 바와 같이 제1 미디어 객체(304)만을 본다. 도 3에서, 와이프 전이 패턴 메쉬가 제1 미디어 객체(304) 상에 매핑되어 제1 매핑된 미디어 객체(304')를 형성한다. 예를 들어, 제1 미디어 객체(304)의 각각의 정점 또는 픽셀점은 3D의 전이 패턴 메쉬 상의 각각의 픽셀점이 2D의 제1 미디어 객체(304) 상의 대응하는 정점 또는 픽셀점과 일치하도록 3D인 전이 패턴 메쉬 상으로 변환 또는 정렬(register)된다. 일 실시예에서, 제2 미디어 객체(306)는 제1 메쉬(106)에 매핑되어 제2 매핑된 미디어 객체(306')를 형성하고, 여기서 제1 메쉬(106)에 대해서는 이하에서 기술된다.

여전히 도 3을 참조하면, 와이프 전이가 진행함에 따라, 제1 매핑된 미디어 객체(304')는 제2 미디어 객체(306')쪽으로 이동하며, 따라서 사용자는 도 3에서 참조 번호(308')로 나타낸 것을 보게 된다. 환언하면, 사용자는 제1 미디어 객체(304)의

제2 미디어 객체(306)로의 전이를 보기 시작한다. 도 3에서, 전이 진행이 완료했을 때, 사용자는 그 결과 얻어진 제2 미디어 객체(306")만을 보게 되며 도 3에 참조 번호(308")로 나타낸 바와 같이, 제1 매핑된 미디어 객체(304")는 사용자에게 보이지 않게 된다.

일 실시예에서, 전이 진행 동안에, 제2 미디어 객체(306')는 제1 매핑된 미디어 객체(304')쪽으로 이동한다. 다른 실시예에서, 제1 매핑된 미디어 객체(304') 및 제2 미디어 객체(306')는 전이 진행 동안에 서로에게로 이동한다. 그 자체로서, 3D의 전이 패턴 메쉬를 사용함으로써, 당업자라면 잘 알고 있지만 2D 전이는 CPU가 아니라 GPU 또는 소프트웨어 애플리케이션에 의해 처리되며, 따라서 이들 전이는 보다 효율적으로 처리된다.

도 1로 돌아가서, 인터페이스(108)는 또한 미디어 객체들 간의 전이가 렌더링될 수 있도록 미디어 객체가 매핑되는 제1 메쉬(106) 및 전이 패턴 메쉬(110)를 수신한다. 예를 들어, 제1 메쉬(106)은 제1 미디어 객체(102)의 정점을 매핑하여 제1 매핑된 미디어 객체를 형성하기 위한 전체 화면 직사각형 메쉬일 수 있다. 환언하면, 제1 미디어 객체(102)의 각각의 정점은 GPU 또는 소프트웨어 애플리케이션이 디스플레이를 위한 각각의 정점에 대응하는 픽셀 값을 처리 또는 계산할 수 있도록 3D 좌표의 제1 메쉬(106)에 매핑된다. 다른 예로서 도 2b를 사용하면, 변환(214)은 1 단위의 폭과 1 단위의 길이를 가지며 1 x 1 정사각형을 생성한다. 도 2b에 나타낸 바와 같이, 변환(214)의 4개의 코너 각각은 2D 좌표계에서 (0,0), (1,0), (1,1) 및 (0,1)로서 표현될 수 있다. 제1 미디어 객체(102)를 제1 메쉬(106)에 매핑함에 있어서, 제1 미디어 객체(102)의 각각의 픽셀점 또는 정점은 변환(214)으로 정렬 또는 변환된다. 또한, 변환의 z좌표는 평탄한 3D 객체를 나타내기 위해 0으로 설정될 수 있다.

일 실시예에서, 제1 메쉬(106)는 텍스처 좌표(예를 들어, 3D의 텍스처 공간에서의 개개의 컬러값에 대응하는 텍스처 값)에 관한 제1 미디어 객체(102)의 정점 데이터를 포함한다. 다른 실시예에서, 각각의 정점에 대한 텍스처 좌표값은 텍스처의 4개의 코너가 제1 메쉬(106)의 4개의 코너에 매핑되도록 x,y 좌표값의 값과 같다(즉, (tu, tv) = (x, y)). 다른 실시예에서, Gouraud 쉐이딩 모드(shading mode)가 사용되고 제1 메쉬(106)를 채우기 위해 텍스처 값이 적용된다. 부록 A의 섹션 2는 제1 메쉬(106)를 생성하는 예시적인 구현에 대해 기술한다. 제1 메쉬 및/또는 전이 패턴 메쉬 구현에 관한 정보에 대해서는, MICROSOFT® 웹 사이트의 MICROSOFT® Developer Network(MSDN®) 서브도메인 상에서 이용가능한 MICROSOFT® Direct3D®의 "DirectX® Software Development Kit"를 참조하기 바라며, 이는 여기에 인용함으로써 그 전체 내용이 본 명세서에 포함된다.

다른 실시예에서, 제1 메쉬(106)는 일련의 정점 및 일련의 다각형면(polygon face)을 정의한다. 제1 메쉬(106) 및 제1 미디어 객체(102)를 수신한 후에, 전이 발생기(112)는 제1 미디어 객체(102)를 제1 메쉬(106)에 매핑하여 제1 매핑된 미디어 객체를 형성한다. 또한, 전이 패턴 메쉬(110)는 제1 미디어 객체(102)로부터 제2 미디어 객체(104)로의 전이를 위해 또는 그 반대를 위해 사용자에 의해 선택되는 전이 패턴들을 포함한다. 예를 들어, 인터페이스(108)는 여러가지 형상을 갖는 와이프 전이, 기타 등등의 다양한 전이의 예들을 디스플레이(120) 상에 디스플레이하고, 사용자는 입력 장치(예를 들어, 마우스 또는 키보드)를 사용하여 원하는 전이 패턴을 선택할 수 있다.

도 4a 내지 도 4c는 본 발명의 일 실시예에 따라 3D 모델을 사용하여 서클 인(circle in) 및 서클 아웃(circle out) 와이프 전이 패턴의 구현을 예시한 것이다. 먼저, 도 4a는 전이 패턴으로서의 서클-아웃 와이프 전이의 측면 투시도를 나타낸 것이다. 들어오는 미디어 객체 또는 제1 미디어 객체(402)(서클 전이 패턴에 의해 매핑된 후)는 나가는 미디어 객체 또는 제2 미디어 객체(418)로 전이하는 것으로 도시되어 있다. 예를 들어, 처음에(또는 z 좌표값이 0 근방일 때), 들어오는 미디어 객체 또는 제1 미디어 객체(402)는 나가는 미디어 객체 또는 제2 미디어 객체(418) 상의 작은 원으로 나타난다. 이 서클 아웃 와이프 전이에서, 사용자는 원(circle)이 화면 또는 디스플레이(120)의 중앙으로부터 나오는 것을 본다. 전이가 진행함에 따라, 이 원은 더 커지게 된다. 들어오는 미디어 객체, 즉 제1 미디어 객체(402)는 원 내에 나타나고 원이 더 커짐에 따라 나가는 미디어 객체, 즉 제2 미디어 객체(418)는 들어오는 미디어 객체, 즉 제1 미디어 객체(402)에 의해 대체된다. 그 자체로서, 들어오는 미디어 객체, 즉 제1 미디어 객체(402)에 대한 z 좌표는 전이 과정을 나타낸다. z 좌표값이 증가함에 따라, 점 또는 작은 원이 작은 반경에서 더 큰 반경으로 증대하게 되고 따라서 나가는 미디어 객체, 즉 제2 미디어 객체(418)는 들어오는 미디어 객체, 즉 제1 미디어 객체(402)로 전이된다. 일 실시예에서, 보는 공간은 보는 사람의 디스플레이(예를 들어, 디스플레이(120))에 의해 제한되며, 그 자체로서 서클 전이의 일부가 디스플레이(120)에 의해 디스플레이되지 않을 수 있기 때문에 그 일부는 잘려질 수 있다.

이와 유사하게, 또하나의 측면 투시도는 들어오는 미디어 객체, 즉 제2 미디어 객체(404) 및 나가는 미디어 객체, 즉 제1 미디어 객체(422)의 서클-인 와이프 전이(circle-in wipe transition)를 보여준다. 시작 시에(또는 z 좌표값이 0 근방에 있을 때) 점에서 시작하는 것이 아니라, 서클-인 와이프 전이는 들어오는 미디어 객체 또는 제2 미디어 객체(404)가 나가는 미디어 객체 또는 제1 미디어 객체(422)로 전이할 때 큰 서클에서 시작하여 점으로 된다.

추가적으로 설명하기 위해, 도 4b는 서클 와이프 전이의 3가지 뷰를 나타낸 것이다. 스크린 샷(406)은 제1 미디어 객체 A가 서클-아웃 와이프로 제2 미디어 객체 B로 전이하는 것의 스냅샷을 나타낸 것이다. 정투상도(408)는 2D 뷰에서의 원을 나타낸 것이고 측면 투시도(410)는 그 원을 원추 형상으로 나타낸 것이다. 그 자체로서, 본 발명의 실시예들은 원 형상의 전이를 모델(200) 등의 3D 모델에 적용하고, 따라서 2D 원이 원추 형상으로서 변환된다. 3D 모델에서 전이 패턴 메쉬(110)를 사용함으로써, 본 발명의 실시예들은 2D 전이가 보다 효율적으로 처리될 수 있도록 2D 전이를 위한 3D 모델에 관한 데이터를 제2 프로세서 또는 제2 일련의 명령어(118)(일괄하여 "제2 프로세서(118)"이라고 함)로 전송한다.

도 4c는 본 발명의 일 실시예에 따른 서클-아웃(circle-out) 및 서클-인(circle-in) 전이의 진행의 다른 예를 나타낸 것이다. 예를 들어, 블록(412)은 서클-아웃 와이프 전이를 통해 나가는 미디어 객체인 제1 미디어 객체(414)가 들어가는 미디어 객체인 제2 미디어 객체(416)로 전이하는 것을 나타낸 것이다. 전이 진행 0%에서, 블록(412)은 제1 미디어 객체(414)만을 보여준다. 진행 25%에서, 서클-아웃 전이가 제2 미디어 객체(416)를 보여주기 시작함에 따라 제2 미디어 객체(416)가 등장하기 시작한다. 진행이 계속되어 100%로 되면, 블록(412)은 2D 서클-아웃 와이프 전이가 어떻게 디스플레이되는지를 보여준다. 이와 유사하게, 블록(420)은 서클-인 와이프 전이를 통해 나가는 미디어 객체인 제1 미디어 객체(414)가 제2 미디어 객체(416)로 전이하는 것을 나타낸 것이다.

다른 패턴을 사용한 와이프 전이의 사용자 경험이 기술 분야에 잘 알려져 있을 수 있지만, GPU 또는 소프트웨어 애플리케이션을 사용하여 3D 전이 패턴을 사용한 2D 전이를 처리하는 것이 보다 효율적이고 또 CPU 처리를 덜 사용한다는 것을 잘 알 것이다.

또한 도 4a 내지 도 4c(예를 들어, 서클-인 및 서클-아웃 와이프 전이)에 도시한 전이 패턴이 제한이 아닌 예로서 도시된 것임을 잘 알 것이다. 예를 들어, 스와이프(swipe), 대각선 하향(diagonal down), 별(star), 휠(wheel), 나비 넥타이(bow tie), 원(circle), 직사각형, 분할(split), 노출 및 삽입(reveal and inset), 홍채(iris), 채워진 V자(filled V), 지그재그, 체커판(checkerboard), 대각선 박스(diagonal box), 다이아몬드, 레이다(radar), 별들(stars), 원들(circles), 열쇠구멍(keyhold), 하트(heart), 스위프(sweep)(-인, -업, -아웃, -다운), 팬(fan)(-업, -아웃, -다운, -인), 눈(eye), 또는 기타 전이 등의 다른 전이 패턴이 본 발명의 정신을 벗어나지 않고 구현될 수 있다. 부가의 예로서, 본 발명의 일 실시예에 따른 예시적인 전이 패턴의 구현들을 나타낸 도 5a 및 도 5b를 참조하기 바란다. 도 5a는 30% 진행에서의 직사각형 전이를 나타낸 것이다. 도 5a는 또한 정투상도 및 측면 사시도를 나타내고 있다. 이와 유사하게, 도 5b는 30% 진행에서의 지그재그 전이, 전이 패턴의 정투상도 및 측면 사시도를 나타내고 있다. 게다가, 부록 A의 섹션 1은 3D 모델을 사용하여 전이 패턴을 생성하는 구현에 대해 기술한다.

제1 매핑된 미디어 객체를 형성하는 것과 유사하게, 전이 발생기(112)는 인터페이스(108)로부터 제2 미디어 객체(104) 및 전이 패턴 메쉬(110)를 수신한 후에, 제2 매핑된 미디어 객체를 생성하기 위해 전이 패턴 메쉬(110)를 제2 미디어 객체(104)의 정점에 매핑한다. 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체는 제1 매핑된 미디어 객체(122) 및 제2 매핑된 미디어 객체(124)로부터 제1 미디어 객체(102)와 제2 미디어 객체(104) 간의 전이를 디스플레이(120) 상에 렌더링하기 위해 제2 프로세서(118)로 전송된다. 예를 들어, 도 4c의 블록(412)은 제2 프로세서 또는 제2 일련의 명령어(118)를 사용하여 제1 매핑된 미디어 객체(414) 및 제2 매핑된 미디어 객체(416)로부터 서클-아웃 전이를 렌더링한 결과를 나타낸 것이다.

일 실시예에서, 시스템(100)은 또한 전이(126)를 렌더링하기 위한 제2 프로세서(118)에 대한 정점, 텍스처, 스텐실 또는 다른 데이터를 일시적으로 저장하기 위해 프레임 버퍼(116)를 포함한다. 예를 들어, 제2 프로세서(118)는 이하의 렌더링 순서로, 즉 (1) 필요한 경우 전이 패턴 메쉬를 마스크로 렌더링하고, (2) 제1 메쉬(106)를 프레임 버퍼(116)로 렌더링하며, (3) 전이 패턴 메쉬(100)를 프레임 버퍼(116)로 렌더링하는 순서로 와이프 전이(wipe transition)를 처리할 수 있다. 예를 들어, 각 단계에서, 제2 프로세서(118)는 이 시퀀스에서의 렌더링 상태를 유지하고, 따라서 그 다음 미디어 객체 또는 전이를 디스플레이하기 전에, 시스템(100)은 적당한 렌더링 상태를 디스플레이 중인 미디어 객체 또는 전이에 적용할 수 있다. 다른 예에서, 프레임 버퍼(116)는 휘발성 저장 매체, 비휘발성 저장 매체 또는 다른 저장 매체 형태의 비디오 메모리일 수 있다.

다른 실시예에서, 제2 프로세서(118)는 전이(126)의 렌더링을 돕기 위해 임시 객체를 생성한다. 예를 들어, 전이(예를 들어, 와이프 전이)의 렌더링 동안에, 제2 프로세서(118)는 가속 래스터화(accelerated rasterization) 및 하드웨어/소프트웨어 정점 처리 등의 부가의 하드웨어 및 소프트웨어 기능에 액세스하기 위한 임시 객체를 생성할 수 있다. 또 다른 실시예에서, 제2 프로세서(118)는 제1 및 전이 패턴 메쉬, 알파 및 디졸브 블렌딩(이하에서 기술함)을 처리하기 위한 부가의 임시 객체를 생성한다. 예를 들어, 제2 프로세서(118)는 제1 메쉬(106), 전이 패턴 메쉬(110), 알파 및 디졸브 블렌딩을 생성 또

는 로드하기 위한 임시 객체를 생성하며 따라서 이들 객체는 하드웨어 또는 소프트웨어로부터의 특정의 기능을 호출 또는 실행할 수 있다. 제2 프로세서(118)가 전이(126) 또는 다른 처리를 렌더링한 후에, 이들 임시 객체는 저장 장치로부터 제거 또는 파괴될 수 있다.

일 실시예에서, 본 발명의 실시예들은 전이와 함께 부가의 효과를 생성한다. 예를 들어, 도 3의 (b) 및 도 3의 (c) 그리고 도 4a 및 도 4b에 도시된 전이는 전이 패턴이 제1 미디어 객체로부터 제2 미디어 객체로 변할 때 전이 패턴의 샤프 엣지(sharp edge)를 나타내고 있다. 도 6은 본 발명의 일 실시예에 따라 전이 패턴의 샤프 엣지를 갖는 제1 미디어 객체와 제2 미디어 객체 간의 2D 전이를 발생하는 방법을 나타낸 플로우차트이다. 처음에, 단계(602)에서, 시스템(100)은 제1 메쉬(예를 들어, 제1 메쉬(102)) 및 전이 패턴 메쉬(예를 들어, 전이 패턴 메쉬(110))를 로드한다. 예를 들어, 인터페이스(108)는 소프트웨어 애플리케이션에 의해 제공되거나 다른 소스로부터 공급되는 전이 패턴들의 컬렉션으로부터 선택한 후에 사용자로부터 전이 패턴 메쉬(예를 들어, 서클-인, 지그재그, 기타 등등)를 수신한다. 단계(604)에서, 시스템(100)은 선택된 2D 전이를 발생하기 위한 3D 모델을 정의한다. 예를 들어, 모델(200) 및 직교 변환(214)을 사용하여, 시스템(100)은 정의된 변환(214)으로서, 0.0에서 1.0으로의 진행에 대한 애니메이션 루프를 정의한다.

단계(606)에서, 인터페이스(108)는 나가는 미디어 객체인 제1 미디어 객체(예를 들어, 제1 미디어 객체(304))를 수신한다. 단계(608)에서, 전이 발생기(112)는 제1 매핑된 미디어 객체를 형성하기 위해 제1 미디어 객체를 제1 메쉬에 매핑한다. 단계(610)에서, 인터페이스(108)는 제2 미디어 객체(예를 들어, 제2 미디어 객체(306))를 수신하고, 단계(612)에서 전이 발생기(112)는 제2 미디어 객체를 인터페이스(108)로부터 선택/수신된 전이 패턴 메쉬에 매핑한다(매핑의 구현의 예에 대한 도 3의 (a) 내지 도 3의 (f)의 설명을 참조할 것). 그 다음에, 단계(614)에서 제2 프로세서(118)는 제1 매핑된 미디어 객체 및 제2 미디어 객체로부터 제1 미디어 객체와 제2 미디어 객체 간의 전이(예를 들어, 전이(126))를 렌더링한다. 예를 들어, 렌더링에 있어서, 제2 프로세서(118)는 먼저, 전이가 일어나기 전에 제1 매핑된 미디어 객체 및 제2 미디어 객체의 정점, 픽셀 및/또는 텍스처에 관한 데이터를 계산한다. 그 다음에, 제2 프로세서(118)는 진행의 함수로서, 즉 제1 미디어 객체(102)를 제2 미디어 객체(104)로 변경함에 있어서 전이(126)가 진행함에 따라 데이터를 계산한다. 데이터의 계산의 완료 시에, 제2 프로세서(118)는 계산된 데이터를 디스플레이(120) 상에 디스플레이될 포맷으로 변환한다. 예를 들어, 디스플레이(120)는 4x3 종횡비를 갖는 컴퓨터 모니터일 수 있으며, 제2 프로세서(118)는 계산된 데이터를 4x3 종횡비를 갖는 디스플레이(120) 상에 전이(126)를 디스플레이하기에 적당하게 변환할 수 있다. 그 결과, 렌더링 후에, 디스플레이(120)는 제1 미디어 객체(102)로부터 제2 미디어 객체(104)로 전이하는 전이(126)를 디스플레이한다. 부록 A의 섹션 3은 샤프 엣지를 생성하는 한 구현을 기술하고 있다.

기술한 바와 같이, 본 발명의 실시예들은 3D 모델을 사용하여 전이를 발생하는 것 이외에, 미디어 객체들 간의 전이에 대한 부가의 효과를 생성한다. 예를 들어, 본 발명의 실시예들은 제1 미디어 객체가 제2 미디어 객체와 교차할 때 전이 동안에 엣지를 부드럽게 할 수 있다.

이전의 기술들에 의해 하드(hard) 또는 샤프(sharp) 엣지를 부드럽게 하기 위해서, 알파 블렌딩 기술이 사용될 수 있다. 예를 들어, 인터페이스(108)는 전이 패턴의 샤프 엣지를 부드럽게 하기 위한 알파 맵(128)을 수신한다. 일 실시예에서, 제2 프로세서(118)는 프리미티브별(재료 알파) 파라미터, 정점별(정점 알파) 파라미터, 픽셀별(텍스처, 프레임 버퍼 및 렌더 타겟 알파) 파라미터, 기타 알파 블렌딩 파라미터 등의 파라미터를 수신 및 처리한다. 알파 블렌딩 기술을 통해, 알파 값은 제1 미디어 객체와 제2 미디어 객체가 교차하는 엣지 근방에서 점차적으로 감소한다. 일 실시예에서, 알파 맵은 알파 값을 저장하기 위해 사용된다.

도 9 및 도 9a는 본 발명의 일 실시예에 따른 알파 맵(900)의 사용의 예를 나타낸 도면이다. 도 9의 (a)의 객체(904)는 30% 전이 진행에서의 대각선 교차 전이를 나타낸 것이고 도 9의 (b)의 객체(906)는 70% 전이 진행에서의 동일한 대각선 교차 전이를 나타낸 것이다. 30% 진행에서, 라인(902)을 따라 있는 알파 값은 전이 진행을 정의하기 위해 선택된다. 예를 들어, 점(922)에서의 알파 값은 도 9의 (a)의 객체(904)에서의 2개의 대응하는 점(912, 914)에 대한 전이 진행값을 정의한다. 이와 유사하게, 점(950)에서의 알파 값은 도 9의 (a)의 객체(904)에서의 대응하는 점(908)에 대한 전이 진행값을 정의한다. 알파 맵(900)에서의 알파 값은 제1 미디어 객체(102) 및 제2 미디어 객체(104)의 렌더링 동안에 하나 이상의 대응하는 점을 나타내는 데 사용될 수 있음을 잘 알 것이다. 또한, 도 9의 (a) 내지 도 9의 (c)가 흑색 및 백색의 음영으로 도시되어 있지만, 본 발명의 범위를 벗어나지 않고 다른 컬러 쉐이딩 및/또는 컬러 텍스처 정도가 사용될 수 있음을 잘 알 것이다.

70% 진행에서, 라인(910)을 따라 있는 알파 값이 전이 진행을 정의하기 위해 선택된다. 예를 들어, 점(924)에서의 알파 값은 도 9의 (b)의 객체(906)에서의 2개의 대응하는 점(916, 918)에 대한 전이 진행값을 정의한다. 이와 유사하게, 점(952)

에서의 알파 값은 도 9의 (b)의 객체(906)에서의 대응하는 점(920)에 대한 전이 진행값을 정의한다. 전이가 진행함에 따라, 전이 진행을 나타내는 다른 라인은 알파 값이 알파 맵(900)의 어디에서 선택되어야 하는지를 나타낸다는 것을 잘 알 것이다.

도 9a는 본 발명의 일 실시예에 따른 알파 맵(926)에서의 경사 반투명 영역(gradient translucent area)을 나타낸다. 알파 맵(926)은 3개의 영역, 즉 불투명 영역(938), 투명 영역(940) 및 제1 미디어 객체(102)와 제2 미디어 객체(104)가 서로 블렌딩되는 경사 반투명 영역(점선 박스로 정의됨)을 포함한다. 알파 맵(926)에서, 점(928)에서의 알파 값은 100% 백색 및 0% 흑색의 컬러 쉐이딩에 대응하는 값들을 포함하는 반면, 점(932)에서의 알파 값은 0% 백색 및 100% 흑색의 컬러 쉐이딩에 대응하는 값들을 포함한다. 알파 맵(926)을 분할하는 대각선(944)은 50% 백색 쉐이딩 및 50% 흑색 쉐이딩을 나타낸다. 5% 폭 영역(점선으로 둘러싸여 있음)은 50% 백색 및 50% 흑색의 컬러 쉐이딩에 대응하는 알파 값이 선택될 수 있는 영역을 나타낸다. 예를 들어, 영역(930, 942) 내의 알파 값은 50% 백색에 대응하는 값을 가질 수 있다. 일 실시예에서, 경사 반투명 영역의 폭은 0% 내지 50%의 범위 내에 있을 수 있다. 예를 들어, 경사 반투명 영역의 0% 폭에는, 경사 반투명 영역이 없으며, 그 결과 알파 맵(926) 내의 알파 값들은 100% 백색 또는 0% 백색에 대응하는 값을 포함한다. 다른 대안으로서, 50% 폭에서, 알파 맵(926) 내의 알파 값은 모두 50% 백색에 대응하는 값을 포함하게 된다. 알파 맵(926)의 경사 반투명 영역의 폭을 변동시키는 것을 정의함으로써, 본 발명의 실시예들은 2차원 전이에서 소프트 엣지 효과를 생성하기 위해 알파 맵(926) 내의 알파 값을 선택할 수 있다. 부록 A의 섹션 5는 알파 맵의 예시적인 구현을 기술한다.

도 7은 본 발명의 일 실시예에 따라 제1 미디어 객체와 제2 미디어 객체 간의 전이의 일부인 부가의 효과를 생성하는 방법을 나타낸 플로우차트이다. 도 7은 알파 블렌딩 기술을 사용하기 위한 본 발명의 대체 실시예에 대해 기술한 것이다. 단계(602 내지 612)는 도 7에도 공통적이라는 것을 잘 알 것이다. 대체 실시예에서, 단계(714)에서 알파 맵이 수신된다. 그 결과, 제2 프로세서(118)는 제1 매핑된 미디어 객체, 제2 매핑된 미디어 객체 및 알파 맵으로부터 제1 미디어 객체(102)와 제2 미디어 객체(104) 간의 전이(126)를 렌더링한다. 이 경우, 제2 프로세서(118)는 상기한 바와 같이 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체를 렌더링하는 것 이외에 알파 맵을 렌더링한다. 예로서 도 4b의 스크린샷(406)을 사용하면, 제1 매핑된 미디어 객체의 텍스처 또는 쉐이딩은 제2 매핑된 미디어 객체의 것과 다르다. 알파 맵으로 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체를 렌더링함에 있어서, 제2 프로세서(118)는 제1 미디어 객체 및 제2 미디어 객체와 교차하는 엣지를 따라 있는 픽셀에 대응하는 알파 값을 알파 맵의 함수로서 계산함으로써 제1 및 제2 매핑된 객체가 교차하는 엣지를 부드럽게 한다. 제2 프로세서(118)가 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체(104)로부터의 데이터를 진행의 함수로서 계산하기 때문에, 계산된 알파 값은 교차 엣지를 따라 있는 픽셀을 대체시키기 위해 적용된다. 제2 프로세서(118)가 픽셀을 적용하는 것을 완료할 때, 제2 프로세서(118)는 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체로부터의 계산된 데이터 이외에 픽셀을 디스플레이하기 위해 디스플레이(120)로 변환한다. 부록 A의 섹션 4는 제1 미디어 객체와 제2 미디어 객체 간의 교차 엣지를 부드럽게 하는 구현에 대해 기술한다.

또 다른 대체 실시예에서, 제1 미디어 객체와 제2 미디어 객체 간의 교차 엣지에 디졸브 블렌딩 기술이 적용된다. 예를 들어, 이 기술은 제2 프로세서(118)에 의한 2번의 렌더링 프로세스를 필요로 한다. 제2 프로세서는 먼저 디졸브된 알파 값을 갖는 디졸브 맵을 발생하기 위해 전이 패턴 메쉬(110)를 렌더링한다. 두번째 렌더링 프로세스에서, 제2 프로세서(118)는 디졸브된 맵으로 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체를 렌더링한다. 환언하면, 알파 맵으로 렌더링하는 것과 유사하게, 제2 프로세서(118)는 제1 매핑된 미디어 객체 및 제2 미디어 객체로 렌더링될 디졸브된 알파 값을 갖는 별도의 디졸브 맵을 발생한다.

다른 실시예에서, 알파 블렌딩 기술 및 디졸브 블렌딩 기술 둘다가 사용될 수 있다. 예를 들어, 알파 맵으로부터의 알파 값은 알파 블렌딩 기술에서와 같이 검색된다. 그 다음에, 디졸브 블렌딩 효과를 내기 위해, 디졸브된 알파 값을 가지며 또 알파 맵으로부터의 알파 값으로 계산된 디졸브 맵이 생성된다. 예를 들어, 이 계산은 ADDSIGNED 연산(예를 들어, ADDSIGNED는 (알파 맵으로부터의) 알파 1 + (디졸브 맵으로부터의) 알파 2를 계산함)을 적용함으로써 행해지고, 결과를 0 내지 1로 제한한다. 다른 실시예에서, 0.25보다 큰 계산된 알파 값을 갖는 픽셀만이 전이의 일부로서 디스플레이된다. 이 예에서, 이들 픽셀이 디스플레이되지 않지만, 계산된 알파 값은 다른 사용을 위해 저장된다(예를 들어, 이들을 스텐실 버퍼에 저장한다). 부록 A의 섹션 8은 엣지 디졸브 기술의 구현에 대해 기술한다.

다른 대안으로서, 일 실시예에서, 단계(718)에서 인터페이스(108)는 제3 미디어 객체를 수신한다. 예를 들어, 알파 블렌딩, 디졸브 블렌딩 또는 소프트 엣지 기술 이외에, 사용자는 제1 미디어 객체 및/또는 제2 미디어 객체의 텍스처 또는 쉐이딩과 다른 텍스처 또는 쉐이딩 데이터로 이루어진 엣지를 생성할 수 있다. 그 자체로서, 단계(720)에서 전이 발생기(112)는 제3 매핑된 미디어 객체를 형성하기 위해 제3 미디어 객체를 전이 패턴 메쉬에 매핑한다. 단계(716)에서 제2 프로세서(118)는 제1 매핑된 미디어 객체, 제2 매핑된 미디어 객체 및 제3 매핑된 미디어 객체로부터 제1 미디어 객체와 제2 미디어 객체 간의 전이를 렌더링한다. 부록 A의 섹션 6은 제3 미디어 객체를 사용하여 전이를 생성하는 구현에 대해 기술한다.

또 다른 실시예에서, 제2 프로세서(118)는 알파 블렌딩, 디졸브 블렌딩, 소프트 엣지, 샤프 엣지, 제3 미디어 객체 블렌딩 또는 상기한 기술들의 조합 등의 기술을 포함하는 전이를 렌더링한다. 예를 들어, 사용자는 어느 픽셀이 종국에 디스플레이되는지를 제어하기 위해 제3 미디어 객체 블렌딩 기술 및 샤프 엣지 기술의 조합에 의해 전이를 생성할 수 있다. 이 예에서, 전이 발생기(112)는 제2 매핑된 미디어 객체를 형성하기 위해 제2 미디어 객체를 매핑하는 것 이외에 제3 매핑된 미디어 객체를 형성하기 위해 제3 미디어 객체를 매핑한다. 그 다음에, 전이 발생기(112)는 (예를 들어, 현재 애니메이션 진행량을 갖는 마이너스 z 좌표 방향에서) 제3 매핑된 미디어 객체 뒤에 제2 매핑된 미디어 객체를 배치한다. 즉, 사용자가 제3 매핑된 미디어 객체의 픽셀만을 볼 수 있도록, 제2 매핑된 미디어 객체의 정점 또는 픽셀값은 제3 매핑된 미디어 객체의 정점 또는 픽셀값 뒤에 매핑된다. 그 다음에, 전이 발생기(112)는 제1 미디어 객체와 제2 미디어 객체가 교차하는 픽셀을 대체시키기 위해 제3 매핑된 미디어 객체를 사용한다. 환언하면, 앞서 기술한 알파 또는 디졸브 블렌딩 기술과는 달리, 제3 미디어 객체 블렌딩 기술은 제1 미디어 객체(102)와 제2 미디어 객체(104)가 교차하는 엣지를 따라 있는 픽셀의 텍스처 값을 대체시키기 위해 제3 미디어 객체를 사용한다. 따라서, 제2 프로세서(118)가 제3 미디어 객체를 사용하여 제1 미디어 객체와 제2 미디어 객체 간의 전이를 렌더링할 때, 사용자는 디스플레이(120) 상에서 제1 미디어 객체와 제2 미디어 객체가 교차하는 엣지를 따라서 제3 미디어 객체를 보게 된다. 일 실시예에서, 제3 매핑된 미디어 객체의 폭은 좁은 상태에서 시작하고 전이(126)의 애니메이션이 진행함에 따라 점점 넓어진다. 부록 A의 섹션 7은 상기한 전이 효과의 구현에 대해 기술한다.

상기한 기술들의 조합을 사용하는 다른 예에서, 소프트 엣지 기술과 제3 미디어 객체 블렌딩 기술의 조합이 사용될 수 있다. 일 실시예에서, 제2 프로세서(118)는 먼저 제2 매핑된 미디어 객체를 렌더링하고 이어서 제3 매핑된 미디어 객체를 렌더링하거나 그 역으로 렌더링한다. 알파 맵의 알파 값은 제3 매핑된 미디어 객체가 제1 미디어 객체 및/또는 제2 미디어 객체와 교차하는 엣지의 픽셀을 블렌딩하는 데 사용된다.

도 8은 컴퓨터(130) 형태의 범용 컴퓨팅 장치의 한 예를 나타낸 것이다. 본 발명의 일 실시예에서, 컴퓨터(130) 등의 컴퓨터는 본 명세서에 예시되고 기술된 다른 도면들에서 사용하기에 적당하다. 컴퓨터(130)는 하나 이상의 프로세서 또는 프로세싱 유닛(132) 및 시스템 메모리(134)를 갖는다. 예시된 실시예에서, 시스템 버스(136)는 시스템 메모리(134)를 비롯한 여러가지 시스템 컴포넌트를 프로세서(132)에 연결시킨다. 버스(136)는 메모리 버스 또는 메모리 콘트롤러, 주변 버스, AGP(accelerated graphics port) 및 다양한 버스 아키텍처 중 임의의 것을 사용하는 프로세서 또는 로컬 버스를 비롯한 몇가지 유형의 버스 구조 중 임의의 것이다. 제한이 아닌 예로서, 이러한 아키텍처로는 ISA(Industry Standard Architecture) 버스, MCA(Micro Channel Architecture) 버스, EISA(Enhanced ISA) 버스, VESA(Video Electronics Standards Association) 로컬 버스, 및 메자닌 버스라고도 하는 PCI(Peripheral Component Interconnect) 버스가 있다.

컴퓨터(130)는 일반적으로 적어도 어떤 형태의 컴퓨터 판독가능 매체를 갖는다. 휘발성 및 비휘발성 매체, 분리형 및 비분리형 매체 둘다를 포함하는 컴퓨터 판독가능 매체는 컴퓨터(130)에 의해 액세스될 수 있는 임의의 이용가능한 매체일 수 있다. 제한이 아닌 예로서, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 포함한다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터 등의 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 포함한다. 예를 들어, 컴퓨터 저장 매체는 RAM, ROM, EEPROM, 플래쉬 메모리 또는 기타 메모리 기술, CD-ROM, DVD 또는 기타 광학 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치 또는 기타 자기 저장 장치, 또는 원하는 정보를 저장하는 데 사용될 수 있고 또 컴퓨터(130)에 의해 액세스될 수 있는 임의의 다른 매체를 포함한다. 통신 매체는 일반적으로 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터를, 반송파 또는 기타 전송 메카니즘 등의 변조된 데이터 신호에 구현하며, 임의의 정보 전달 매체를 포함한다. 당업자라면 그의 특성 중 하나 이상이 정보를 그 신호에 인코딩하는 방식으로 설정되거나 변경된 변조된 데이터 신호를 잘 알고 있다. 통신 매체의 예로는 유선 네트워크나 직접 유선 연결 등의 유선 매체 또는 음향, RF, 적외선 및 기타 무선 매체 등의 무선 매체가 있다. 상기한 것 중 임의의 것의 조합도 역시 컴퓨터 판독가능 매체의 범위 내에 포함된다.

시스템 메모리(134)는 분리형 및/또는 비분리형, 휘발성 및/또는 비휘발성 메모리 형태의 컴퓨터 저장 매체를 포함한다. 예시된 실시예에서, 시스템 메모리(134)는 판독 전용 메모리(ROM)(138) 및 랜덤 액세스 메모리(RAM)(140)를 포함한다. 시동 중과 같은 때에 컴퓨터(130) 내의 구성요소들 간의 정보 전송을 돕는 기본적인 루틴을 포함하는 기본 입/출력 시스템(142)(BIOS)은 일반적으로 ROM(138)에 저장되어 있다. RAM(140)은 일반적으로 프로세싱 유닛(132)에 의해 즉각 액세스가능하고 및/또는 현재 처리되고 있는 데이터 및/또는 프로그램 모듈을 가지고 있다. 제한이 아닌 예로서, 도 8은 오퍼레이팅 시스템(144), 애플리케이션 프로그램(146), 기타 프로그램 모듈(148), 및 프로그램 데이터(150)를 예시하고 있다.

컴퓨터(130)는 또한 기타의 분리형/비분리형, 휘발성/비휘발성 컴퓨터 저장 매체를 포함할 수 있다. 예를 들어, 도 8은 비분리형 비휘발성 자기 매체로부터 판독하거나 그에 기록하는 하드 디스크 드라이브(154)를 나타내고 있다. 도 8은 또한 분리형 비휘발성 자기 디스크(158)로부터 판독하거나 그에 기록하는 자기 디스크 드라이브(156), 및 CD-ROM 또는 기타 광

학 매체 등의 분리형 비휘발성 광학 디스크(162)로부터 판독하거나 그에 기록하는 광학 디스크 드라이브(160)를 예시하고 있다. 예시적인 오퍼레이팅 환경에서 사용될 수 있는 기타의 분리형/비분리형, 휘발성/비휘발성 컴퓨터 저장 매체는 자기 테이프 카세트, 플래쉬 메모리 카드, DVD, 디지털 비디오 테이프, 고체 상태 RAM, 고체 상태 ROM, 및 기타 등등을 포함하지만, 이에 한정되는 것은 아니다. 하드 디스크 드라이브(154), 자기 디스크 드라이브(156) 및 광학 디스크 드라이브(160)는 일반적으로 인터페이스(166) 등의 비휘발성 메모리 인터페이스에 의해 시스템 버스(136)에 연결되어 있다.

이상에서 기술하고 도 8에 예시된 드라이브 또는 다른 대용량 저장 장치 및 그의 관련 컴퓨터 저장 매체는 컴퓨터(130)의 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 및 기타 데이터의 저장을 제공한다. 예를 들어, 도 8에서, 하드 디스크 드라이브(154)는 오퍼레이팅 시스템(170), 애플리케이션 프로그램(172), 기타 프로그램 모듈(174) 및 프로그램 데이터(176)를 저장하는 것으로서 예시되어 있다. 유의할 점은 이들 컴포넌트가 오퍼레이팅 시스템(144), 애플리케이션 프로그램(146), 기타 프로그램 모듈(148) 및 프로그램 데이터(150)과 동일한 것이거나 다른 것일 수 있다는 것이다. 오퍼레이팅 시스템(170), 애플리케이션 프로그램(172), 기타 프로그램 모듈(174) 및 프로그램 데이터(176)는 최소한 이들이 서로 다른 복사본임을 예시하기 위해 여기에서는 서로 다른 참조 번호가 부기되어 있다.

사용자는 키보드(180) 및 포인팅 장치(182)(예를 들어, 마우스, 트랙볼 또는 터치 패드) 등의 입력 장치 또는 사용자 인터페이스 선택 장치를 통해 컴퓨터(130)에 명령 및 정보를 입력할 수 있다. 다른 입력 장치(도시 생략)는 마이크로폰, 조이스틱, 게임 패드, 위성 안테나, 스캐너, 또는 기타 등등을 포함할 수 있다. 이들 및 다른 입력 장치는 시스템 버스(136)에 연결된 사용자 입력 인터페이스(184)를 통해 프로세싱 유닛(132)에 연결되어 있지만, 병렬 포트, 게임 포트 또는 USB(Universal Serial Bus) 등의 다른 인터페이스 및 버스 구조에 의해 연결될 수 있다. 모니터(188) 또는 기타의 유형의 디스플레이 장치도 역시 비디오 인터페이스(190) 등의 인터페이스를 통해 시스템 버스(136)에 연결되어 있다. 모니터(188) 이외에, 컴퓨터는 종종 출력 주변 인터페이스(도시 생략)를 통해 연결될 수 있는 프린터 및 스피커 등의 기타의 주변 출력 장치(도시 생략)를 포함한다.

컴퓨터(130)는 원격 컴퓨터(194) 등의 하나 이상의 원격 컴퓨터로의 논리적 연결을 사용하여 네트워크화된 환경에서 동작할 수 있다. 원격 컴퓨터(194)는 퍼스널 컴퓨터, 서버, 라우터, 네트워크 PC, 피어 장치 또는 기타의 통상적인 네트워크 노드일 수 있으며, 일반적으로 컴퓨터(130)와 관련하여 상기한 구성요소들 대부분 또는 그 전부를 포함한다. 도 8에 도시된 논리적 연결은 근거리 통신망(LAN)(196) 및 원거리 통신망(WAN)(198)을 포함하지만, 기타의 네트워크도 포함할 수 있다. LAN(196) 및/또는 WAN(198)은 유선 네트워크, 무선 네트워크, 이들의 조합, 기타 등등일 수 있다. 이러한 네트워킹 환경은 사무실, 전사적 컴퓨터 네트워크, 인트라넷 및 글로벌 컴퓨터 네트워크(예를 들어, 인터넷)에서 통상적인 것이다.

근거리 통신망 환경에서 사용될 때, 컴퓨터(130)는 네트워크 인터페이스 또는 어댑터(186)를 통해 LAN(196)에 연결된다. 원거리 통신망 환경에서 사용될 때, 컴퓨터(130)는 일반적으로 인터넷 등의 WAN(198)을 통한 통신을 설정하는 모뎀(178) 또는 기타의 수단을 포함한다. 내장형 또는 외장형일 수 있는 모뎀(178)은 사용자 입력 인터페이스(184) 또는 기타의 적절한 메카니즘을 통해 시스템 버스(136)에 연결된다. 네트워크화된 환경에서, 컴퓨터(130)와 관련하여 도시된 프로그램 모듈 또는 그의 일부분은 원격 메모리 저장 장치(도시 생략)에 저장될 수 있다. 제한이 아닌 예로서, 도 8은 원격 애플리케이션 프로그램(192)을 메모리 장치 상에 존재하는 것으로 예시하고 있다. 도시된 네트워크 연결이 예시적인 것이고 또 컴퓨터들 간의 통신 링크를 설정하는 기타의 수단이 사용될 수 있다.

일반적으로, 컴퓨터(130)의 데이터 프로세서는 컴퓨터의 여러가지 컴퓨터 판독가능 저장 매체에 서로 다른 때에 저장된 명령어에 의해 프로그램된다. 프로그램 및 오퍼레이팅 시스템은 일반적으로 예를 들어 플로피 디스크 또는 CD-ROM 상으로 배포된다. 그곳로부터, 이들은 컴퓨터의 2차 메모리로 설치 또는 로드된다. 실행 시에, 이들은 적어도 부분적으로 컴퓨터의 1차 전자 메모리로 로드된다. 본 명세서에 기술된 본 발명은 이들 및 다른 여러가지 유형의 컴퓨터 판독가능 저장 매체를 포함하며, 이러한 매체는 마이크로프로세서 또는 다른 데이터 프로세서와 관련하여 이하에 기술하는 단계들을 구현하는 명령어 또는 프로그램을 가지고 있다. 본 발명은 또한 본 명세서에 기술된 방법 및 기술에 따라 프로그램된 컴퓨터 자체도 포함한다.

설명의 목적상, 오퍼레이팅 시스템 등의 프로그램 및 다른 실행가능 프로그램 컴포넌트가 본 명세서에서 개별적인 블록으로 도시되어 있다. 그렇지만, 이러한 프로그램 및 컴포넌트가 다양한 때에 컴퓨터의 서로 다른 저장 컴포넌트에 존재하고 컴퓨터의 데이터 프로세서(들)에 의해 실행된다는 것을 잘 알 것이다.

컴퓨터(130)를 포함하는 예시적인 컴퓨팅 시스템 환경과 관련하여 기술되어 있지만, 본 발명은 여러가지 다른 범용 또는 전용 컴퓨팅 시스템 환경 또는 구성에서 동작한다. 이 컴퓨팅 시스템 환경은 본 발명의 사용 또는 기능의 범위에 관하여 어떤 제한을 암시하려는 것이 아니다. 게다가, 이 컴퓨팅 시스템 환경은 예시적인 오퍼레이팅 환경에 도시된 컴포넌트 중 임의의 하나 또는 이들의 조합에 관하여 어떤 의존관계 또는 요구사항을 갖는 것으로 해석되어서도 안된다. 본 발명과 함께

사용하기에 적합할 수 있는 공지의 컴퓨팅 시스템, 환경 및/또는 구성의 예로는 퍼스널 컴퓨터, 서버 컴퓨터, 핸드-헬드 또는 랩톱 장치, 멀티프로세서 시스템, 마이크로프로세서 기반 시스템, 셋톱 박스, 프로그램가능 가전 제품, 이동 전화, 네트워크 PC, 미니컴퓨터, 메인프레임 컴퓨터, 상기 시스템 또는 장치 중 임의의 것을 포함하는 분산 컴퓨팅 환경, 기타 등등이 있지만, 이에 한정되는 것은 아니다.

본 발명은 하나 이상의 컴퓨터 또는 다른 장치에 의해 실행되는, 프로그램 모듈 등의 컴퓨터 실행가능 명령어의 일반적인 관점에서 기술될 수 있다. 일반적으로, 프로그램 모듈은 특정의 작업을 수행하거나 특정의 추상 데이터 유형을 구현하는 루틴, 프로그램, 객체, 컴포넌트, 및 데이터 구조를 포함하지만 이에 한정되는 것은 아니다. 본 발명은 또한 작업들이 통신 네트워크를 통해 연결되어 있는 원격 프로세싱 장치에 의해 수행되는 분산 컴퓨팅 환경에서 실시될 수 있다. 분산 컴퓨팅 환경에서, 프로그램 모듈은 메모리 저장 장치를 비롯한 로컬 및 원격 컴퓨터 저장 매체 둘다에 위치할 수 있다.

소프트웨어 아키텍처의 관점에서의 인터페이스는 소프트웨어 모듈, 컴포넌트, 코드 부분, 또는 다른 컴퓨터 실행가능 명령어 시퀀스를 포함한다. 인터페이스는 예를 들어 제1 모듈을 대신하여 컴퓨팅 작업을 수행하기 위해 제2 모듈에 액세스하는 제1 모듈을 포함한다. 한 예에서, 제1 및 제2 모듈은 오퍼레이팅 시스템에 의해 제공되는 것 등의 애플리케이션 프로그램 인터페이스(API), 컴포넌트 객체 모델(COM)(예를 들어, 피어-투-피어 애플리케이션 통신을 위한 것임), 및 확장가능 마크업 언어 메타데이터 교환 포맷(extensible markup language metadata interchange format, XMI) 인터페이스(예를 들어, 웹 서비스 간의 통신을 위한 것임)를 포함한다.

인터페이스는 J2EE(Java 2 Platform Enterprise Edition), COM, 또는 DCOM(Distributed COM) 예 등의 밀결합 동기 구현(tightly coupled, synchronous implmentation)일 수 있다. 다른 대안으로서 또는 그에 부가하여, 인터페이스는 웹 서비스(예를 들어, SOAP(simple object access protocol)을 사용함) 등에서와 같이 소결합 비동기 구현(loosely coupled, asynchronous implmentation)일 수 있다. 일반적으로, 인터페이스는 이하의 특성, 즉 밀결합, 소결합, 동기 및 비동기의 임의의 조합을 포함한다. 게다가, 인터페이스는 표준 프로토콜, 독점 프로토콜, 또는 표준 및 독점 프로토콜의 임의의 조합에 부합될 수 있다.

본 명세서에서 기술되는 인터페이스는 모두 단일의 인터페이스의 일부이거나 별개의 인터페이스로서 또는 이들의 임의의 조합으로서 구현될 수 있다. 인터페이스는 기능을 제공하기 위해 로컬적으로 또는 원격적으로 실행될 수 있다. 게다가, 인터페이스는 본 명세서 예시되거나 기술된 것보다 더 많은 또는 더 적은 기능을 포함할 수 있다.

발명의 효과

본 명세서에 예시되고 기술된 방법들의 실행 또는 수행 순서는 달리 언급하지 않는 한, 필요불가결한 것이 아니다. 즉, 본 방법의 요소들은 달리 언급하지 않는 한, 임의의 순서로 수행될 수 있으며 또 이 방법은 본 명세서에 개시된 것보다 더 많거나 더 적은 구성요소를 포함할 수 있다. 예를 들어, 다른 요소 이전에, 그와 동시에, 또는 그 이후에 특정의 요소를 실행 또는 수행하는 것이 본 발명의 범위 내에 속하는 것으로 생각된다.

본 발명의 구성요소 또는 그의 실시예(들)을 소개할 때, 이, 그, 저 등의 지시사는 하나 이상의 구성요소가 있음을 의미하기 위한 것이다. 용어 "구비하는", "포함하는" 및 "갖는"은 포함적인 것을 말하기 위함이며 열거된 구성요소 이외에 부가의 구성요소가 있을 수 있음을 의미한다.

이상의 것을 살펴보면, 본 발명의 몇가지 목적이 달성되고 기타의 유익한 결과가 얻어짐을 알 수 있다.

본 발명의 범위를 벗어나지 않고 상기 시스템 및 방법에 여러가지 변경이 행해질 수 있기 때문에, 상기 설명에 포함되고 첨부 도면에 도시된 모든 개시 내용은 예시적인 것이며 제한적이 아닌 것으로 해석되어야만 한다.

부록 A

섹션 1: 지오메트리 발생 샘플 코드

발생하고자 하는 "삽입" 와이프 패턴에 기초하여, 단면적 25%, 50% 및 75%에서의 전이 패턴을 얻는다. 이들을 겹쳐 놓고 정점을 연결하면, 이와 같은 모델을 얻을 수 있다. x는 출력의 폭이고 y는 출력의 높이이며, z는 전이의 진행이다.

```
const DWORD VERTEX_XYZ_TEX2::FVF = D3DFVF_XYZ | D3DFVF_TEX2;
```

```
m_dwNumPatternIndices = 6;
m_dwNumPatternVertices = 4;
D3DXCreateMeshFVF( // 이하의 것을 갖는 D3DXMeshobject를
생성하는 DirectX API
m_dwNumPatternIndices / 3, // 면의 수 / 삼각형
m_dwNumPatternVertices, // 정점의 수
D3DXMESH_SYSTEMMEM, // 시스템 메모리에 D3DXMesh를 생성함
VERTEX_XYZ_TEX2:: FVF, // 정점 포맷: XYZ 위치 및
2 세트 텍스처 좌표
pd3dDevice, // GPU 프로세스의 인스턴스
&pSysMemMesh // D3DXMESH 포인터를 반환
);
VERTEX_XYZ_TEX2* pVertices = NULL;
pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pVertices );
// D3DXMesh 객체로부터 VertexBuffer를 가져옴
// 상기 드로잉에 기초하여 하드 코딩된 (x, y, z) 좌표로 정점을 채움
pVertices[O].v.x = 1.0f;
pVertices[O].v.y = 1.0f;
pVertices[O].v.z = 1.0f;
// (t0.x, t0.y) = (x, y) 및 (tl.x, tl.y) = (z, 진행)으로 함
pVertices[O].t0.x = pVertices[O].v.x;
pVertices[O].t0.y = pVertices[O].v.y;
pVertices[0].tl.x = pVertices[O].v.z;
pVertices[0].t1.y = 0.50f; // 진행
pVertices[l].v.x = 0.0f;
pVertices[l].v.y = 1.0f;
```

```
pVertices[1].v.z = 0.0f;
pVertices[1].t0.x = pVertices[1] .v.x;
pVertices[1].t0.y = pVertices[1] .v.y;
pVertices[1].tl.x = pVertices[1] .v.z;
pVertices[1].tl.y = 0.50f; // 진행
pVertices[2].v.x = 0.0f;
pVertices[2].v.y = 0.0f;
pVertices[2].v.z = 1.0f;
pVertices[2].t0.x = pVertices[2] .v.x;
pVertices[2].t0.y = pVertices[2] .v.y;
pVertices[2].tl.x = pVertices[2] .v.z;
pVertices[2].tl.y = 0.50f; // 진행
pVertices[3].v.x = 1.0f;
pVertices[3].v.y = 0.0f;
pVertices[3].v.z = 1.0f;
pVertices[3].t0.x = pVertices[3] .v.x;
pVertices[3].t0.y = pVertices[3] .v.y;
pVertices[3].tl.x = pVertices[3] .v.z;
pVertices[3].tl.y = 0.50f; // 진행
pSysMemMesh->UnlockVertexBuffer() ;
// Direct3D에게 이 메쉬 객체가 스왑 아웃될 수 있음을 통지함, 이는 현재 사용되지 않음
WORD* pIndices = NULL;
pSysMemMesh->LockIndexBuffer( D3DLOCK_NOSYSLOCK,
(void**)&pIndices);
// D3DXMesh 객체로부터 인덱스 버퍼를 가져옴
// 인덱스를 채움(시계 방향으로).
// 삼각형 1
```

```
pIndices[0] = 0;

pIndices[1] = 1;

pIndices[2] = 3;

// 삼각형 2

pIndices[3] = 1;

pIndices[4J = 2;

pIndices[5] = 3;

pSysMemMesh->UnlockIndexBuffer();

// 시스템 메모리 및 장치 메모리 내의 D3DXMesh 객체를 관리하기 위한 CDXUTMesh 객체를 생성

*ppMesh = new CDXUTMesh( L"CInsetPatternMesh" );

(*ppMesh)->m_pSysMemMesh = pSysMemMesh.Detach();

// 이전에 생성된 D3DXMesh 객체를 CDXUTMesh에 의해 관리함
```

섹션 2: 메쉬 발생 샘플 코드

메쉬는 DXUTMesh 객체로서 생성된다.

각각의 DXUTMesh 객체는 2개의 D3DXMesh 객체를 관리한다 - 하나는 시스템 메모리(CPU가 액세스함)에 있고 다른 하나는 장치 메모리(GPU/디스플레이 어댑터가 액세스함)에 있음. 시스템 메모리 내의 D3DXMesh 객체에 변경이 행해질 때마다, 애플리케이션은 그를 장치 메모리로 다운로드해야만 한다. 그렇지 않으면, GPU는 모든 드로잉 명령에 대해 D3DXMesh 객체를 재사용하게 된다. 양 D3DXMesh 객체의 콘텐츠는 동일하다. 이는 하나의 정점 버퍼와 하나의 인덱스 버퍼를 포함한다.

정점 버퍼는 정점의 어레이이다. 예를 들어, 와이프 전이는 어느 데이터가 정점에 포함되어 있는지를 정의한다. 각각의 정점은 이하에 주어진 예 등의 3개의 엔트리를 갖는다.

3D 좌표,(x, y, z)는 정점 위치에 대한 것이다. 각각의 정점 위치는 원, 스타, 하트, 기타 등등의 기하 형상에 기초하여 계산된다.

2D 텍스처 좌표, (tul, tvl)는 비디오 소스 상의 어디에서 정점에 대한 적색/녹색/청색(RGB) 값을 찾아야 하는지를 GPU 또는 일련의 명령어에 지시를 하기 위한 것이다. 와이프 전이 예에서, tul은 x 값에 할당되고, tv1은 y 값에 할당된다.

제2 2D 텍스처 좌표, (tu2, tv2)는 알파맵 상의 어디에서 이 정점에 대한 알파 값을 찾아야 하는지를 GPU 또는 일련의 명령어에 지시를 하기 위한 것이다. tu2는 z 값에 할당되고, tv2는 런타임/애니메이션 타임 시에 진행 % 값에 할당된다.

인덱스 버퍼는 정수 인덱스의 어레이이다. 각각의 인덱스는 정점 버퍼에 인덱싱되어 있다. 매 3개의 인덱스가 삼각형을 형성한다. 모든 삼각형은 하나의 3D 모델을 형성한다. Alias의 Maya® 등의 다른 3D 모델링 도구가 .x 파일을 생성하는 데 사용될 수 있다. 생성된 .x 파일을 로드할 때, 가시 영역이 0.0 및 1.0 범위에 맞도록 3D 모델을 스케일링함으로써 각각의 (x, y, z) 좌표의 정규화가 필요할 수 있다.

섹션 3: 샤프 엣지 샘플 코드

```
// Direct3D 상태를 설정함
// pd3dDevice는 Direct3D / GPU의 인스턴스 포인터이다.
pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE, FALSE); // 알파 테스트를 수행할 필요가 없음
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE,FALSE); // 알파 블렌딩을 수행할 필요가 없음
pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, FALSE); // 스텐실 테스트를 수행할 필요가 없음
pd3dDevice->SetRenderState( D3DRS_ZENABLE, D3DZB_TRUE); // 깊이(Depth) 테스트를 인에이블함
pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE,TRUE); // Z 값을 깊이(Depth) 버퍼에 저장함
// Direct3D 텍스처 블렌딩 상태를 설정함
// Direct3D는 최대 8개 스테이지를 허용하지만 우리는 1개만 필요함
pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0);
// 스테이지 0에 대해 t0 텍스처 좌표를 사용함
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP SELECTARGl);
// RGB 값을 출력함
pd3dDevice->SetTextureStageState( 0,
D3DTSS_COLORARGl, D3DTA_TEXTURE );
// 텍스처로부터 RGB 값을 출력함
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
// 스테이지 0 이상으로부터 알파 출력 없음
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP DISABLE );
//스테이지 1 이상으로부터의 RGB 출력 없음
// 들어오는 소스를 드로잉함
// 텍스처 2 및 제1 메쉬를 사용함
pd3dDevice->SetTexture(0, pTextureIn);
// pTextureIn은 들어오는 비디오 프레임을 저장하는 텍스처 객체이다.
// 이 텍스처를 텍스처 블렌딩 스테이지 0로 설정함
```

LPD3DXMESH pPlaneMesh = m_pPlaneMesh->GetLocalMesh(); // 장치/GPU 메모리에 저장되어 있는 D3DXMesh 객체를 CDXUTMesh로부터 가져옴.

pPlaneMesh->DrawSubset(0); // D3D/GPU가 이 PlaneMesh, TextureIn 및 현재 설정을 사용하여 드로잉을 실행함

// 나가는 소스를 드로잉함

// 텍스처 1 및 전이 패턴 메쉬를 사용함

// 전이 패턴 메쉬를 z 방향으로 아래로 이동시키기 위해 이동 변환 매트릭스를 적용함

D3DXMATRIX matTranslation;

D3DXMatrixTranslation(&matTranslation, 0, 0, -fltProgress);

// PatternMesh를 z 방향으로 아래로 이동시키기 위한 이동 변환 매트릭스를 생성함

pd3dDevice->SetTransform(D3DTS_WORLD, &matTranslation);

// D3D 실세계 변환 매트릭스를 설정함

// D3D 실세계 변환은 모델 공간 좌표를 실세계 공간으로 변환함

// 전이 패턴 메쉬에서의 (0, 0, 0)은 (0, 0, -fltProgress)로 된다

pd3dDevice->SetTexture(O, pTextureOut);

LPD3DXMESH_pPatternMesh = m_pPatternMesh->GetLocalMesh();

pPatternMesh->DrawSubset(0); // D3D/GPU가 이 PatternMesh, TextureOut 및 현재 설정을 사용하여 드로잉을 실행하게 함

이 루틴은 각 프레임에 대해 반복된다. fltProgress는 전이 기간, 프레임 레이트 및 현재 프레임에 기초하여 계산된다.

D3D/GPU가 제1 메쉬/들어오는 비디오 소스를 드로잉할 때, 이는 전체 TextureIn을 렌더 타겟 표면에 갖다 대고 또 z 값을 렌더 타겟 표면에서의 각 픽셀에 대한 깊이 버퍼로 설정함

D3D/GPU가 전이 패턴 메쉬/나가는 비디오 소스를 드로잉할 때, 이는 깊이 테스트를 수행한다. 이는 전이 패턴 메쉬 픽셀의 z 값을 깊이 버퍼 내의 그 픽셀의 z 값과 비교한다. 보는 사람으로부터 멀리 있는 픽셀이 버려진다.

섹션 4: 소프트 엣지 샘플 코드

// 들어오는 비디오 소스를 먼저 드로잉함

// Direct3D 상태를 설정함

// pd3dDevice는 Direct3D/GPU의 인스턴스 포인터이다.

pd3dDevice->SetRenderState(D3DRS_SHADEMODE, D3DSHADE_GOURAUD);

// 매끄러운 쉐이딩 모드로 설정됨. 따라서, D3D / GPU는 각각의 정점에서의 텍스처 좌표를 계산한 다음에 각각의 삼각형의 3개의 정점 간의 보간을 수행하여 그 삼각형 내의 각 픽셀의 텍스처 좌표를 계산한다. 그 텍스처 좌표는 나중에 선택된 텍스처로부터 RGB 및 알파 값을 찾는 데 사용된다.

pd3dDevice->SetRenderState(D3DRS_ALPHATESTENABLE, FALSE); // 알파 테스트를 수행할 필요가 없음

pd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE,FALSE); // 알파 블렌딩을 수행할 필요가 없음

pd3dDevice->SetRenderState(D3DRS_STENCILENABLE, FALSE); // 스텐실 테스트를 할 필요가 없음

pd3dDevice->SetRenderState(D3DRS_ZENABLE, D3DZB_FALSE); // 깊이 테스트를 할 필요가 없음

pd3dDevice->SetRenderState(D3DRS_ZWRITEENABLE, FALSE) ; // 깊이 버퍼에 Z 값을 저장할 필요가 없음

// Direct3D 텍스처 블렌딩 상태를 설정

// Direct3D는 최대 8 스테이지를 허용하지만 우리는 1개만 필요하다.

// pTextureln의 (tul, tvl)로부터 각 정점에 대한 RGB 값을 찾음

pd3dDevice->SetTextureStageState(0, D3DTSS_TEXCOORDINDEX , 0);

// 스테이지 0에 대한 tul, tv1 텍스처 코드를 사용함

pd3dDevice->SetTextureStageState(0, D3DTSS_COLOROP,

D3DTOP_SELECTARGl); // RGB 값을 출력함

pd3dDevice->SetTextureStageState(0, D3DTSS_COLORARG1,

D3DTA_TEXTURE); // 텍스처로부터 RGB 값을 출력함

pd3dDevice->SetTextureStageState(0, D3DTSS_ALPHAOP,

D3DTOP DISABLE); // 스테이지 0 이상으로부터 알파 출력 없음

pd3dDevice->SetTextureStageState(1, D3DTSS_COLOROP,

D3DTOP_DISABLE);

// 스테이지 0 이상으로부터 RGB 출력 없음

// 들어오는 소스를 드로잉함

// 텍스처 2 및 제1 메쉬를 사용함

pd3dDevice->SetTexture(O, pTextureIn);

// pTextureIn은 들어오는 비디오 프레임을 저장하는 텍스처 객체이다

// 이 텍스처를 텍스처 블렌딩 스테이지 0으로 설정함

LPD3DXMESH pPlaneMesh D3DXMesh = m_pPlaneMesh->GetLocalMesh(); // 장치/GPU 메모리에 저장된 D3DXMesh 객체를 CDXUTMesh로부터 가져옴

pPlaneMesh->DrawSubset(0); // D3D/GPU가 이 PlaneMesh, TextureIn 및 현재 설정을 사용하여 드로잉을 실행하게 함

```
// 나가는 소스를 드로잉함
//렌더 상태를 설정함
// 알파 블렌딩 동작에 대해 설정함
pd3dDevice->SetRenderState( D3DRS ALPHABLENDENABLE, TRUE);
pd3dDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND_SRCALPHA );
pd3dDevice->SetRenderState( D3DRS_DESTBLEND, DBLEND_INVSRCALPHA);
// 텍스처 블렌딩
// 스테이지 0:
// ColorOp : 텍스처 선택
// AlphaOp : 현재
// pTextureOut의 (tul, tvl)로부터 각 정점에 대한 RGB 값을 찾음
pd3dDevice->SetTextureStageState( 0, D3DTSS TEXCOORDINDEX , 0 );
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP, D3DTOP_SELECTARGl);
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARGl,
D3DTA_TEXTURE );
// 알파 값을 설정하지 않음
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_SELECTARGl);
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARGl,
D3DTA CURRENT );
// 스테이지 1
// ColorOp : 현재
// AlphaOp : 텍스처 선택
RTN_HR_IF_FAILED(pd3dDevice->SetTexture(l, m_pAlphaMapTexture));
// 이전의 스테이지로부터의 RGB 값의 변동 없음
pd3dDevice->SetTextureStageState( 1, D3DTSS TEXCOORDINDEX , 1 );
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
```

```
D3DTOP_SELECTARGl);
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLORARGl,
D3DTA_CURRENT );
// pAlphaMapTexture의 (tu2, tv2)로부터 각 정점에 대한 알파 값을 찾음
pd3dDevice->SetTextureStageState( I, D3DTSS_ALPHAOP, D3DTOP SELECTARGl);
pd3dDevice->SetTextureStageState( I, D3DTSS_ALPHAARGl,
D3DTA TEXTURE );
// 스테이지 2 : 디스에이블
pd3dDevice->SetTextureStageState( 2, D3DTSS_ALPHAOP,
D3DTOP_DISABLE) ;
pd3dDevice->SetTextureStageState( 2, D3DTSS_COLOROP,
D3DTOP DISABLE );
pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR
) ;
pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR) ;
pd3dDevice->SetSamplerState( 1, D3DSAMP_MINFILTER,
D3DTEXF LINEAR);
pd3dDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER,
D3DTEXF_LINEAR)j
pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP) ;
pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSV,
D3DTADDRESS CLAMP );
// 전이 패턴 메쉬의 각 정점의 tv2를 진행( Progress)으로 설정함
VERTEX_XYZ_TEX2* pVertices = NULL;
pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
```

(void**)&pVertices);

DWORD dwNumPatternVertices = pSysMemMesh->GetNumVertices();

for(DWORD i = O; i < dwNumPatternVertices; i++)

pVertices [i].t2.y = fltProgress; // 진행

// fltProgress는 전이 기간, 프레임 레이트 및 현재 프레임에 기초하여 계산됨

pSysMemMesh->UnlockVertexBuffer();

m_pMesh->InvalidateDeviceObjects();

m_pMesh->RestoreDeviceObjects(pd3dDevice) ;

// RestoreDeviceObjects()는 로컬 메쉬를 생성함

// 텍스처 1 및 전이 패턴 메쉬를 사용함

pd3dDevice->SetTexture(O, pTextureOut);

// pTextureOut은 나가는 비디오 프레임을 저장하는 텍스처 객체임

// 이 텍스처를 텍스처 블렌딩 스테이지 0으로 설정함

LPD3DXMESH pPatternMesh = m_pPatternMesh->GetLocalMesh();

pPatternMesh->DrawSubset(0); // D3D/GPU가 이 PatternMesh, TextureOut 및 현재 설정을 사용하여 드로잉하게 함

전술한 루틴은 각 프레임에 대해 반복된다. D3D/GPU가 제1 메쉬/들어오는 비디오 소스를 드로잉할 때, 이는 전체 TextureIn을 렌더 타겟 표면에 갖다 댄다. D3D / GPU가 전이 패턴 메쉬/나가는 비디오 소스를 드로잉할 때, 이는 알파맵으로부터 알파 값을 찾고 알파 블렌딩 계산에서 렌더 타겟에서의 TextureOut 및 RGB 값과 함께 이를 사용한다.

섹션 5: 알파맵 발생 샘플 코드

각각의 정점은 위치 및 컬러 정보를 포함한다.

우측-상부 코너 (1,0)은 불투명하다. 우측-하부 코너 (0,1)은 투명하다. 좌측-상부로부터 우측-하부로의 라인은 50% 투명하다. 투명에서 불투명으로의 구배는 사인 곡선을 따른다. 이 메쉬를 텍스처로 렌더링하고 알파맵을 사용하여 소프트 엣지를 생성한다. 번 엣지(Burn Edge)에 대한 알파맵은 동일한 형상을 갖지만 다른 컬러/알파 값을 갖는다. 우측-상부 코너 (1,0) 및 우측-하부 코너(0,1)는 투명하다. 좌측-상부로부터 우측-하부로의 라인은 번 엣지 세기에 의해 설정된 투명도를 갖는다.

struct ALPHAMAPVERTEX

{

D3DXVECTOR3 v; // 위치

D3DCOLOR color; // 컬러

```
static const DWORD FVF;
} ;
const DWORD ALPHAMAPVERTEX::FVF = D3DFVF_XYZ | D3DFVF DIFFUSE;
m_wNumSegments = 36;
m dwNumVertice = 2 // 좌측-상부 및 우측-하부
+ (m_wNumSegments + 1) * 3; // 3: 좌측-중간-우측
m_dwNumIndices
= 2 * 4
+ (m_wNumSegments - 1) * 6;
m dwNumIndices *= 3;
D3DXCreateMeshFVF(
m_dwNumIndices / 3, // 면의 수
m_dwNumVertices, // 정점의 수
D3DXMESH_SYSTEMMEM, // 시스템 메모리에 저장
ALPHAMAPVERTEX::FVF, // 정점 포맷
pd3dDevice, // GPU 인스턴스
&pSysMemMesh // LPD3DXMESH를 반환
) ;
// 기록을 위해 VertexBuffer를 로크함
ALPHAMAPVERTEX* pVertices = NULL;
pSysMemMesh->LockVertexBuffer( D3DLOCK_NOSYSLOCK,
(void**) &pVertices) ;
D3DCOLOR clrRight;
D3DCOLOR clrMiddle;
D3DCOLOR clrLeft;
if(m_Type == SoftEdge)
{
```

```
clrLeft = 0;
clrMiddle = Ox80808080;
clrRight = Oxffffffff;
}
else if(m_Type == BurnEdge)
{
clrLeft = 0;
clrMiddle = D3DCOLOR COLORVALUE
(m_fltBurnlntensity,m_fltBurnlnte~sitYlm_f1tBurnlntensity,m_fltBu rnlntensity) ;
clrRight = 0;
}
else
{
clrLeft = Oxffffffff;
clrMiddle = 0;
clrRight = Oxftfftfft;
}
// 좌측-하부
pVertices[O}.v.x = 0.0f;
pVertices[O].v.y = 1.0f;
pVertices[O].v.z = 0.0f;
pVertices[O].color = clrLeft;
// 우측-상부(소프트 엣지의 경우 100%; 번 엣지의 경우 0)
pVertices[l].v.x = 1.0f;
pVertices[l].v.y = 0.0f;
pVertices[l].v.z = 0.0f;
pVertices[l].color = clrRight;
```

```
//좌측-상부
pVertices[2].v.x = 0.0f;
pVertices[2].v.y = 0.0f;
pVertices[2].v.z = 0.0f;
pVertices[2].color = clrLeft;
pVertices[3].v.x = 0.0f;
pVertices[3].v.y = 0.0f;
pVertices[3].v.z = 0.0f;
pVertices[3].color = clrMiddle;
pVertices[4].v.x = 0.0f;
pVertices[4].v.y = 0.0f;
pVertices[4].v.z = 0.0f;
pVertices[4].color = clrRight;
// 우측 하부
pVertices[5].v.x = 1.0f;
pVertices[5].v.y = 1.0f;
pVertices[5].v.z = 0.0f;
pVertices[5].color = clrLeft;
pVertices[6].v.x = 1.0f;
pVertices[6].v.y = 1.0f;
pVertices[6].v.z = 0.0f;
pVertices[6].color = clrMiddle;
pVertices[7].v.x = 1.0f;
pVertices[7].v.y = 1.0f;
pVertices[7].v.z = 0.0f;
pVertices[7].color = clrRight;
WORD index = 8; // 위에서 0 내지 7이 할당되어 있다.
```

```
// 0 내지 180도의 사인 곡선을 사용함; 사인값은 0에서 1로 되었다가 0으로 됨
WORD i;
for(i = 1; i < m_wNumSegments; i+ + )
{
FLOAT fltXY = (FLOAT)i / (FLOAT)m_wNumSegments;
FLOAT fltAngle = D3DX_PI * fltXY;
FLOAT fltDistance = ::sin(fltAngle);
fltDistance *= m fltGradientWidth I 2.0f;
// 구배 영역의 폭을 조정
// fltDistance 범위를 검사
if(fltXY < 0.5f)
// 상부 절반, 그러면 fltXY - fltDistance > 0
fltDistance = min(fltXY, fltoistance);
else if(fltXY > 0.5f)
// 하부 절반, 그러면 fltXY +  fltDistance < 1
fltDistance = rnin(1.0f - fltXY,
fl tDistance) ;
else
// 중간, 그러면 fltXY +  fltDistance < 1이고
fltXY- fltDistance > 0
fltDistance = min (0.5f, fltDistance) ;
// 좌측
pVertices[index].v.x = fltXY - fltDistance;
pVertices[index].v.y = fltXY;
pVertices[index].v.z = 0.0f
pVertices[index].color = clrLeft;
// 중간
```

```
pVertices[index +  1].v.x = fltXY;

pVertices[index +  1].v.y = fltXY;

pVertices[index +  1].v.z = O.Of;

pVertices[index +  1].color = clrMiddle;

// 우측

pVertices[index +  2].v.x = fltXY +

fltDistance;

pVertices[index +  2].v.y = fltXY ;

pVertices[index +  2].v.z = O.Of;

pVertices[index +  2].color = clrRight;

index + = 3;

}

RTN_HR_IF_FAILED( pSysMemMesh->UnlockVertexBuffer());

// 기록을 위해 IndexBuffer를 로크함

WORD* pIndices = NULL;

RTN_HR_IF_FAILED( pSysMemMesh-

>LockIndexBuffer(D3DLOCK_NOSYSLOCK, (void**)&pIndices));

RTN_HR_IF_BADPTR(pIndices);

// 인덱스를 채움

// 최좌측

pIndices[O] = 2;

pIndices[1] = 8;

pIndices[2] = 0;

// 좌측 중앙

pIndices[3] = 3;

pIndices[4] = 9;

pIndices[5] = 8;
```

```
// 우측 중앙
pIndices[6] = 3;
pIndices[7] = 10;
pIndices[8] = 9;
// 상부 스캔 라인이 모두 불투명하도록 최상부에서 마지막까지 드로잉함
index = 9;
for(i = 1; l < m_wNumSegments - 1; i ++ )
{
// 좌측까지
pIndices[index + 0] = 0;
pIndices[index + 1] = i * 3 + 5;
pIndices[index + 2] = i * 3 + 8;
// 좌측 사분면
pIndices[index + 3] = l * 3 + 5;
pIndices[index + 4] = i * 3 + 6;
pIndices[index + 5] = i * 3 + 8;
pIndices[index + 6] = i * 3 + 6;
pIndices[index + 7] = i * 3 + 9;
pIndices[index + 8] = i * 3 + 8;
// 우측 사분면
pIndices[index + 9] = i * 3 + 6;
pIndices[index + 10] = i * 3 + 7;
pIndices[index + 11] = i * 3 + 9;
pIndices[index + 12] = 1 * 3 + 7;
pIndices[index + 13] = i * 3 + 10;
pIndices[index + 14] = i * 3 + 9;
// 우측까지
```

```
pIndices[index + 15] = 1;
pIndices[index + 16] = i * 3 + 10;
pIndices[index + 17] = i * 3 + 7;
index + = 18 ;
}
// 최우측
pIndices[index + 0] = 7;
pIndices[index + 1] = i * 3 + 7;
pIndices[index + 2] = 1;
// 우측 중앙
pIndices[index + 3] = 6;
pIndices[index + 4] = i * 3 + 6;
pIndices[index + 5] = i * 3 + 7;
// 좌측 중앙
pIndices[index + 6] = 6;
pIndices[index + 7] = i * 3 + 5;
pIndices[index + 8] = i * 3 + 6;
// 최하부
pIndices[index + 9] = 5;
pIndices[index + 10] = 0;
pIndices[index + 11] = i * 3 + 5;
// 최상부
pIndices[index + 12] = 4 ;
pIndices[index + 13] =1;
pIndices[index + 14] = 10;
}
RTN_HR_IF_FAILEO( pSysMemMesh->UnlockIndexBuffer());
```

```
// 시스템 메모리 및 장치 메모리 내의 D3DXMesh 객체를 관리하기 위해 CDXUTMesh를 생성함
m_pMesh = new CDXUTMesh( L"CAlphaMap" );
RTN_HR_IF_BADNEW(m_pMesh) ;
m_pMesh->m_pSysMemMesh = pSysMemMesh.Detach() ;
// 이전에 생성된 D3DXMesh 객체를 CDXUTMesh에 의해 관리함.
m_pMesh->RestoreDeviceObjects(pd3dDevice);
// 장치 메모리에 복사본을 생성함
#ifdef _DEBUG
// 메쉬가 .X 파일로 저장될 수 있다.
if (get_MeshFilename( ))
{
VERIFY_OK(SaveMesh(pd3dDevice, get_MeshFilename( )));
}
#endif
// 첫번째 스캔 라인이 불투명하고
// 마지막 스캔 라인이 모두 투명하도록
// 128 x 256 알파맵을 사용
CComPtr<IDirect3DSurface9> pSurfaceLevel;
pd3dDevice->CreateRenderTarget(
128, 256, // UINT Width, UINT Height,
D3DFMT_A8R8G8B8, // 렌더 타겟 포맷
D3DMULTISAMPLE_NONE, //D3DMULTISAMPLE_TYPE,
D3DMULTISAMPLE_NONE, // 멀티샘플 품질
TRUE, //BOOL Lockable,
&pSurfaceLevel, // IDirect3DSurface9**을 반환함
ppSurface,
NULL // HANDLE* pSharedHandle
```

```
) ;
// D3D/GPU가 이 표면까지 렌더링하게 함
pd3dDevice->SetRenderTarget(O, pSurfaceLevel);
// D3D 상태를 설정함
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
pd3dDevice->SetRenderState( D3DRS_SRCBLEND, D3DBLEND SRCALPHA
) ;
pd3dDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND ZERO );
// 기존의 픽셀을 블렌딩할 필요없음
pd3dDevice->SetRenderState(D3DRS_SHADEMOOE, D3DSHADE_GOURAUD);
// 매끄러운 쉐이딩
pd3dDevice->SetRenderState(D3DRS_COLORWRITEENABLE,
D3DCOLORWRITEENABLE RED D3DCOLORWRITEENABLE GREEN
| D3DCOLORWRITEENABLE_BLUE |
D3DCOLORWRITEENABLE ALPHA );
D3DXMATRIX matWorld;
D3DXMatrixIdentity( &matWorld );
pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld ); // 스케일/이동/회전 없음
// D3D 카메라 설정
D3DXMATRIX matView;
D3DXVECTOR3 vFromPt = D3DXVECTOR3( 0.5f, 0.5f, 10.Of );
D3DXVECTOR3 vLookatpt = D3DXVECTOR3( 0.5f, 0.5f, O.Of);
D3DXVECTOR3 vUpVec = D3DXVECTOR3( O.Of, -1.0f, O.Of );
D3DXMatrixLookAtLH( &matView, &vFromPt, &vLookatPt, &vUpVec );
pd3dDevice->SetTransform( D3DTS_VIEW, &matView);
D3DXMATRIX matproj;
D3DXMatrixOrthoLH( &matProj, 1.0f, 1.0f, 0.01f 100.Of );
```

```
pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
// D3D 텍스처 블렌딩 스테이지 상태를 설정함
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP SELECTARG1 );
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARGl,
D3DTA_DIFFUSE) ;
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_SELECTARGI );
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARGl,
D3DTA_DIFFUSE);
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_DISABLE );
pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
D3DTOP DISABLE );
RTN_HR_IF_BADNEW(m_pMesh);
LPD3DXMESH pMesh = m_pMesh->GetLocalMesh();
RTN_HR_IF_BADNEW(pMesh);
// 뷰포트를 클리어함
pd3dDevice->Clear( OL, NULL, D3DCLEAR_TARGET, OxOOOOOOOO,
1.0f, OL);
// 장면을 시작함
pd3dDevice->BeginScene();
HRESULT hr = S_OK;
// 드로잉을 실행함
pMesh->DrawSubset( 0 ));
// 장면을 종료
pd3dDevice->EndScene();
```

```
// 알파맵을 저장하기 위한 텍스처를 생성함
CComPtr<IDirect3DTexture9> pTexture;
if(S_OK != pd3dDevice->CreateTexture(
128, 256, // UINT Width, UINT Height,
1. // MIP Levels,
D3DUSAGE_DYNAMIC,
D3DFMT_A8, // 텍스처 포맷, 알파 전용
D3DPOOL_DEFAULT, // 장치 메모리에 넣음
&pTexture, // 텍스처를 반환함
NULL) // HANDLE* pSharedHandle) ||
|| pTexture == NULL)
{
// A8 포맷 텍스처를 생성하지 못함
// A8R8GBB8 포맷을 시도함. 더 많은 메모리를 사용함
pd3dDevice->CreateTexture(
128, 256,
1,
D3DUSAGE_DYNAMIC,
D3DFMT_A8R8G8B8,
D3DPOOL_DEFAULT,
&pTexture,
NULL
) ;
}
CComPtr<IDirect3DSurface9> pDestSurfaceLevel;
pTexture->GetSurfaceLevel(0/* 첫번째 MIP 레벨*/,
&pDestSurfaceLevel) ;
```

```
// 렌더 타겟 픽셀을 텍스처로 복사함
D3DXLoadSurfaceFromSurface(
pDestSurfaceLevel, // LPDIRECT3DSURFACE9 pDestSurface,
NULL, // CONST PALETTEENTRY *pDestPalette,
NULL, // CONST RECT *pDestRect,
pSurfaceLevel, // LPDIRECT3DSURFACE9 pSrcSurface,
NULL, // CONST PALETTEENTRY *pSrcPalette,
NULL, // CONST RECT *pSrcRect,
D3DX FILTER_POINT, // DWORD Filter,
0 // D3DCOLOR ColorKey
) ;
*ppTexture = pTexture.Detach();
// 이 텍스처는 알파맵임
#ifdef_DEBUG
// 알파맵이 .dds 또는 jpg 파일로 저장될 수 있음
D3DXSaveSurfaceToFile(
_T(".\\Shared\\RuntimeAlphaMap.dds"), // 파일 이름
D3DXIFF_DDS, // 파일 포맷
pDestSurfaceLevel, // 이 표면을 파일로 저장함
NULL, // NO PALETTEENTRY,
NULL // NO RECT; 전체 표면
) ;
#endif
```

섹션 6: 번 엣지 샘플 코드

```
// 샤프 또는 소프트 엣지 기술로 와이프 전이를 렌더링한 후에, 번 엣지 알파맵으로 팬시 엣지 텍스처를 전이 패턴 메쉬로 렌더링한다. 번 엣지 알파맵은 소프트 엣지 기술의 알파맵과 동일한 형상 정점 배열을 갖는다. 그렇지만, 컬러 배열은 다르다. 좌측-상부에서 우측-하부로의 라인의 양측에 투명하다. 좌측-상부에서 우측 하부로의 라인의 알파값은 BurnIntensity이다. 중앙 라인으로부터, 알파값은 서서히 0으로 감소된다.
```

```
// 렌더 상태를 설정
pd3dDevice->SetRenderState( D3DRS_SHADEMODE, D3DSHADE_GOURAUD);
pd3dDevice->SetRenderState( D3DRS_ZENABLE, D3DZB_FALSE);
pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE, FALSE);
pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE, FALSE);
pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, FALSE);
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, TRUE );
pd3dDevice->SetRenderState( D3DRS_SRCBLEND,D3DBLEND_SRCALPHA );
pd3dDevice->SetRenderState(D3DRS_DESTBLEND,D3DBLEND_INVSRCALPHA);
if(m_enumFancyEdge != FancyEdgeColor && m_enumFancyEdge !=
FancyEdgeColors)
{
pd3dDevice->SetRenderState( D3DRS_DESTBLEND, D3DBLEND_ONE );
// 따라서 팬시 엣지 텍스트 내의 흑색 픽셀이 나타나지 않음
}
// 텍스처 블렌딩
// 스테이지 0
// ColorOp : 텍스처 선택
// AlphaOp : 현재(Current)
pd3dDevice->SetTextureStageState( 0, D3DTSS TEXCOORDINDEX , 0 );
pd3dDevice->SetTextureStageState( 0,
D3DTSS_COLOROP, D3DTOP_SELECTARGI );
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARGl,
D3DTA_TEXTURE );
if (m_enumFancyEdge == FancyEdgeColor ) || m_enumFancyEdge ==
FancyEdgeColors)
{
```

```
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARGl,
D3DTA_TFACTOR );
// RenderState TextureFactor로부터 RGB 값을 가져옴
}
pd3dDevice->SetTextureStageState( 0,
D3DTSS_ALPHAOP, D3DTOP SELECTARGI );
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARGl,
D3DTA_CURRENT );
// 스테이지 1 :
// ColorOp : 현재
// AlphaOp : 텍스처 선택
pd3dDevice->SetTexture(1, m_pAlphaMapTexture);
pd3dDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX , 1 );
pd3dDevice->SetTextureStageState( 1,
D3DTSS_COLOROP, D3DTOP_SELECTARG1 );
pd3dDevice->SetTextureStageState( 1, D3 DTSS_COLORARG 1 ,
D3DTA_CURRENT );
pd3dDevice->SetTextureStageState( 1,
D3DTSS_ALPHAOP, D3DTOP_SELECTARG1 );
pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAARGl,
D3DTA TEXTURE );
// 스테이지 2 : 디스에이블
pd3dDevice->SetTextureStageState( 2,
D3DTSS_ALPHAOP, D3DTOP_DISABLE );
pd3dDevice->SetTextureStageState( 2,
D3DTSS_COLOROP, D3DTOP DISABLE );
pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR
```

```
) ;
pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR
) ;
pd3dDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR
) ;
pd3dDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR
) ;
pd3dDevice->SetSamplerState(l, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP );
pd3dDevice->SetSamplerState(l, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );
if(m_enumFancyEdge == FancyEdgeColor || m_enumFancyEdge ==
FancyEdgeColors)
{
if (m_enumFancyEdge == FancyEdgeColors)
{
D3DXCOLOR clrStart( m_colorFancyEdgeStart );
D3DXCOLOR clrStop( m_colorFancyEdgeStop );
D3DXCOLOR clrOut( (DWORD)O );
D3DXColorLerp(&clrOut, &clrStart, &clrStop,
(FLOAT)fTime) ;
D3DCOLOR clr =
D3DCOLOR_COLORVALUE(clrOut.r,clrOut.g,clrOut.b,clrOut.a
) ;
pd3dDevice->SetRenderState( D3DRS_TEXTUREFACTOR, clr );
}
else
```

```
{
pd3dDevice->SetRenderState{ D3DRS_TEXTUREFACTOR,
m_colorFancyEdgeStart );
}
else
{
pd3dDevice->SetTexture(O, m_pFancyEdgeTexture);
}
pPatternMesh->DrawSubset( 0 ); // D3D/GPU가 현재 설정으로 이 PatternMesh, TextureOut를 사용하여 드로잉을 실행하게 함
```

섹션 7: 팬시 엣지 샘플 코드

```
FLOAT fltShift = m_fltGradientWidth;
if(m_fltGradientWidth == O.Of)
{
fltShift = 0.05f;
}
fltProgress = (FLOAT) fTime;
if (m_fProgressShift)
{
if(fltProgress > fltShift)
{
fltProgress = fltShift +  fltProgress;
}
else
{
fltProgress *= 2.0f;
}
```

}

일 실시예에서, 전이 패턴 메쉬는 샤프/소프트/디졸브 엣지 기술과 정확히 동일한 프로세스를 통해 2번 렌더링된다. 첫번째 패스는 불꽃 이미지 또는 파도 이미지일 수 있는 팬시 엣지 텍스처를 렌더링한다. 두번째 패스는 상기 소스 코드로 진행이 시프트된 나가는 비디오 소스를 렌더링한다.

섹션 8: 디졸브 엣지 샘플 코드

// 들어오는 비디오 소스를 먼저 드로잉함

// Direct3D 상태를 설정함

// pd3dDevice는 Direct3D / GPU의 인스턴스 포인터이다

pd3dDevice->SetRenderState(D3DRS_ALPHATESTENABLE, FALSE); // 알파 테스트를 수행할 필요없음

pd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE,FALSE); // 알파 블렌딩을 수행할 필요없음

pd3dDevice->SetRenderState(D3DRS_STENCILENABLE, FALSE); // 스텐실 테스트를 수행할 필요없음

pd3dDevice->SetRenderState(D3DRS_ZENABLE, D3DZB_FALSE) ; // 깊이 테스트 필요없음

pd3dDevice->SetRenderState(D3DRS_ZWRITEENABLE, FALSE) ; // Z값을 깊이 버퍼에 저장할 필요없음

// Direct3D 텍스처 블렌딩 상태를 설정함

// Direct3D는 최대 8개의 스테이지를 허용하지만 우리는 1개만 필요하다

// pTextureIn의 (tu1, tv1)로부터 각 정점에 대한 RGB 값을 찾음

pd3dDevice->SetTextureStageState(0, D3DTSS_TEXCOORDINDEX , 0);

// 스테이지 0에 대해 t0 텍스처 좌표를 사용함

pd3dDevice->SetTextureStageState(0, D3DTSS_COLOROP,

D3DTOP_SELECTARGl); // RGB 값을 출력함

pd3dDevice->SetTextureStageState(0, D3DTSS_COLORARGl,

D3DTA_TEXTURE); // 텍스처로부터의 RGB 값을 출력함

pd3dDevice->SetTextureStageState(0, D3DTSS_ALPHAOP,

D3DTOP_DISABLE);

// 스테이지 0 이상으로부터 알파 출력 없음

pd3dDevice->SetTextureStageState(1, D3DTSS_COLOROP,

D3DTOP_DISABLE);

// 스테이지 1 이상으로부터 RGB 출력 없음

// 들어오는 소스를 드로잉함

// 텍스처 2 및 제1 메쉬를 사용함

pd3dDevice->SetTexture(0, pTextureIn);

// pTextureIn은 들어오는 비디오 프레임을 저장하는 텍스처 객체임

// 이 텍스처를 텍스처 블렌딩 스테이지 0으로 설정함

LPD3DXMESH pPlaneMesh = m_pPlaneMesh->GetLocalMesh(); // CDXUTMesh로부터 장치/GPU 메모리에 저장된 D3DXMesh객체를 가져옴.

pPlaneMesh->DrawSubset(0); // D3D/GPU가 이 PlaneMesh, TextureIn 및 현재 설정을 사용하여 드로잉을 실행하게 함

// 나가는 소스를 드로잉함

// 전이 패턴 메쉬의 각 정점의 tv2를 진행(Progress)으로 설정함

VERTEX_XYZ_TEX2* pVertices = NULL;

pSysMemMesh->LockVertexBuffer(D3DLOCK_NOSYSLOCK, (void**)&pVertices);

DWORD dwNumPatternVertices = pSysMemMesh->GetNumVertices();

for(DWORD i = 0; i < dwNumPatternVertices; i++)

pVertices[i].t2.y = fltProgress; // 진행

// fltProgress는 전이 기간, 프레임 레이트, 및 현재 프레임에 기초하여 계산됨

pSysMemMesh->UnlockVertexBuffer();

m_pMesh->InvalidateDeviceObjects();

m_pMesh->RestoreDeviceObjects(pd3dDevice);

// RestoreDeviceObjects()는 로컬 메쉬를 생성함

// 디졸브 효과 기술은 patternMesh를 2번 렌더링할 것을 요한다. 첫번째 패스에서, 이는 디졸브맵 및 알파맵으로부터의 알파값을 가산한다. 이 값은 알파 테스트 동작 중에 사전 정의된 문턱값과 비교된다. 픽셀에 대해 증가된 스텐실 값은 이 테스트를 통과한다. RGB 값은 렌더 타겟에 기록되지 않는다.

// 컬러 버퍼로의 기록을 방지하기 위해 src = 0 및 dest = 1 상태에서 알파 블렌딩을 사용함

pd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);

pd3dDevice->SetRenderState(D3DRS_SRCBLEND,

D3DBLEND _ ZERO);

pd3dDevice->SetRenderState(D3DRS_DESTBLEND,

```
D3DBLEND_ONE) ;
// 깊이 버퍼링을 디스에이블함
pd3dDevice->SetRenderState{ D3DRS_ZENABLE, D3DZB_FALSE) ;
pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE, FALSE) ;
// 알파 테스트를 통과하는 픽셀의 스텐실을 증가시키도록 스텐실 함수를 설정함
pd3dDevice->SetRenderState( D3DRS_STENCILPASS,
D3DSTENCILOP_INCR) ;
pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, TRUE);
pd3dDevice->SetRenderState( D3DRS_STENCILFUNC, D3DCMP_ALWAYS) ;
// 알파 테스트를 설정함
pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE,TRUE);
pd3dDevice->SetRenderState( D3DRS_ALPHAFUNC,D3DCMP_GREATER);
pd3dDevice->SetRenderState( D3DRS_ALPHAREF, Ox40);
// 텍스처 블렌딩
// 스테이지 0 :
// ColorOp : 현재
// AlphaOp : 텍스처 선택
RTN_HR_IF_FAILED(pd3dDevice->SetTexture(O, m_pAlphaMapTexture));
pd3dDevice->SetTextureStageState( 0, D3DTSS TEXCOORDINDEX , 1 ) ;
pd3dDevice->SetTextureStageState( 0, D3DTSS COLOROP ,
D3DTOP_SELECTARG1);
pd3dDevice->SetTextureStageState( 0, 03DTSS_COLORARGl,
D3DTA_CURRENT ) ;
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAOP,
D3DTOP_SELECTARGl
) ;
pd3dDevice->SetTextureStageState( 0, D3DTSS_ALPHAARGl,
```

```
D3DTA_TEXTURE ) ;
// 스테이지 1
// ColorOp : 현재
// AlphaOp : Add - Signed
RTN_HR_IF_FAILED(pd3dDevice->SetTexture(1,
m_pDissolveMapTexture));
pd3dDevice->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX , 0 );
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_SELECTARG1);
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLORARG1,
D3DTA_CURRENT );
// 2개의 알파 텍스처로부터의 알파값을 가산함
pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
D3DTOP_ADDSIGNED);
pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAARG1,
D3DTA_TEXTURE );
// 스테이지 2 : 디스에이블
pd3dDevice->SetTextureStageState( 2, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
pd3dDevice->SetTextureStageState( 2, D3DTSS_COLOROP,
D3DTOP_DISABLE );
pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR
) ;
pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR
) ;
pd3dDevice->SetSamplerState( 1, D3DSAMP_MINFILTER, D3DTEXF_POINT
) ;
```

```
pd3dDevice->SetSamplerState( 1, D3DSAMP_MAGFILTER, D3DTEXF_POINT
) ;
pd3dDevice->SetSamplerState(O, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP );
pd3dDevice->SetSamplerState(O, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );
pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSU,
D3DTADDRESS_CLAMP );
pd3dDevice->SetSamplerState(1, D3DSAMP_ADDRESSV,
D3DTADDRESS_CLAMP );
LPD3DXMESH pPatternMesh = m_pPatternMesh->GetLocalMesh();
pPatternMesh->DrawSubset( 0 ); // D3D/GPU가 이 현재 설정으로 PatternMesh를 사용하여 드로잉을 실행하게 함
// 두번째 패스, 이는 깊이 테스트 및 스텐실 테스트를 수행함
// 양 테스트를 통과하는 픽셀만이 렌더 타겟으로 드로잉된다
// 스텐실 테스트는 와이프 엣지를 따라 랜덤 패턴을 형성한다
// 깊이 테스트는 픽셀이 엣지를 넘어서 팝업하지 않도록 보장하기 위한 것임
// 비디오 1을 렌더링하도록 렌더링 상태를 설정함
// 알파 블렌딩 및 테스트를 디스에이블함
pd3dDevice->SetRenderState( D3DRS_ALPHATESTENABLE, FALSE) ;
pd3dDevice->SetRenderState( D3DRS_ALPHABLENDENABLE, FALSE) ;
// 첫번째 장면의 렌더링을 위한 스텐실 상태를 설정함
pd3dDevice->SetRenderState( D3DRS_STENCILENABLE, TRUE) ;
pd3dDevice->SetRenderState( D3DRS_STENCILPASS,
D3DSTENCILOP_KEEP) ;
pd3dDevlce->SetRenderState( D3DRS_STENCILREF, 1) ;
pd3dDevice->SetRenderState( D3DRS_STENCILFUNC, D3DCMP_EQUAL) ;
pd3dDevice->SetRenderState( D3DRS_ZENABLE, D3DZB_TRUE);
```

```
pd3dDevice->SetRenderState( D3DRS_ZWRITEENABLE, TRUE);
pd3dDevice->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX , 0);
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLOROP,
D3DTOP_SELECTARGl);
pd3dDevice->SetTextureStageState( 0, D3DTSS_COLORARGl,
D3DTA_TEXTURE );
pd3dDevice->SetTextureStageState( 1, D3DTSS_ALPHAOP,
D3DTOP_DISABLE );
pd3dDevice->SetTextureStageState( 1, D3DTSS_COLOROP,
D3DTOP_DISABLE );
pd3dDevice->SetSamplerState( 0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR
);
pd3dDevice->SetSamplerState( 0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR
);
// 텍스처 1 및 전이 패턴 메쉬를 사용함
pd3dDevice->SetTexture(0, pTextureOut);
// pTextureOut은 나가는 비디오 프레임을 저장하는 텍스처 객체임
// 이 텍스처를 텍스처 블렌딩 스테이지 0으로 설정함
// 전이 패턴 메쉬를 z 방향으로 아래로 이동시키기 위해 이동 변환 매트릭스를 적용함
D3DXMATRIX matTranslation;
D3DXMatrixTranslation( &matTranslation, 0, 0, -fltProgress +
m_fltGradientWidth);
// 패턴 메쉬를 z 방향에서 아래로 이동시키기 위해 이동 변환 매트릭스를생성함
pd3dDevice->SetTransform( D3DTS_WORLD, &matTranslation );
// D3D 실세계 변환 매트릭스를 설정함
// D3D 실세계 변환은 모델 공간 좌표를 실세계 공간으로 변환함
// 전이 패턴 메쉬에서의 (0, 0, 0)은 (0, 0, -fltProgress +  m_fltGradientWidth)가 된다
```

pPatternMesh->DrawSubset(0); // D3D/GPU가 현재 설정으로 이 PatternMesh, TextureOut를 사용하여 드로잉을 실행하게 함

(57) 청구의 범위

청구항 1.

그래픽 처리 유닛을 사용하여 제1 미디어 객체와 제2 미디어 객체 간의 2차원 전이를 발생하기 위한 방법으로서,

원하는 2차원 전이 패턴에 대응하는 3차원 모델을 정의하는 단계, 및

상기 그래픽 처리 유닛을 사용하여 상기 정의된 3차원 모델의 함수로서 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 전이를 발생하는 단계를 포함하는 방법.

청구항 2.

제1항에 있어서, 제1 매핑된 미디어 객체를 생성하기 위해 2차원 좌표계에서의 상기 제1 미디어 객체의 복수의 정점을 제1 메쉬에 매핑하는 단계 - 상기 제1 메쉬는 상기 3차원 모델과 관련이 있음 -,

제2 매핑된 미디어 객체를 생성하기 위해 상기 제2 미디어 객체의 복수의 정점을 전이 패턴 메쉬에 매핑하는 단계 - 상기 전이 패턴 메쉬는 상기 3차원 모델과 관련이 있음 -, 및

상기 그래픽 처리 유닛을 사용하여 상기 제1 매핑된 미디어 객체 및 상기 제2 매핑된 미디어 객체로부터 상기 전이를 렌더링하는 단계를 더 포함하는 방법.

청구항 3.

제2항에 있어서, 알파 값을 포함하는 알파 맵을 발생하는 단계를 더 포함하며,

상기 렌더링하는 단계는 상기 제1 매핑된 미디어 객체 및 상기 제2 매핑된 미디어 객체로부터 상기 전이 및 상기 알파 맵을 상기 알파 맵의 런타임 발생의 함수로서 렌더링하는 단계를 포함하는 방법.

청구항 4.

제3항에 있어서, 상기 제1 미디어 객체 및 상기 제2 미디어 객체와 교차하는 엣지를 따라 있는 상기 픽셀에 대응하는 알파 값을 상기 알파 맵의 함수로서 계산하는 단계, 및

상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하는 단계와 협동하여 상기 픽셀에 상기 계산된 알파 값을 적용하는 단계를 더 포함하는 방법.

청구항 5.

제4항에 있어서, 상기 제1 미디어 객체 및 상기 제2 미디어 객체와 교차하는 엣지를 따라 있는 상기 픽셀에 대응하는 디졸브된 알파 값을 디졸브 맵(dissolve map)의 함수로서 계산하는 단계, 및

상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하는 단계와 협동하여 상기 픽셀에 상기 계산된 디졸브된 알파 값을 적용하는 단계를 더 포함하는 방법.

청구항 6.

제1항에 있어서, 상기 전이 패턴 메쉬는 3차원 입방체(volume cube)를 나타내며,

상기 3차원 입방체는 x, y 및 z 좌표 파라미터를 포함하는 방법.

청구항 7.

제1항에 있어서, 제3 매핑된 미디어 객체를 생성하기 위해 제3 미디어 객체의 복수의 정점을 상기 수신된 전이 패턴 메쉬에 매핑하는 단계, 및

상기 제1 매핑된 미디어 객체, 상기 제2 매핑된 미디어 객체 및 상기 제3 매핑된 미디어 객체로부터 상기 전이를 렌더링하는 단계를 더 포함하는 방법.

청구항 8.

제1항의 방법을 수행하는 컴퓨터 실행가능 명령어를 갖는 하나 이상의 컴퓨터 판독가능 매체.

청구항 9.

제1 미디어 객체와 제2 미디어 객체 간의 2차원 전이를 발생하기 위한 시스템으로서,

상기 제1 미디어 객체, 상기 제2 미디어 객체, 및 전이 패턴 메쉬를 수신하기 위한 인터페이스,

제1 매핑된 미디어 객체를 생성하기 위해 상기 수신된 제1 미디어 객체의 복수의 정점을 제1 메쉬에 매핑하기 위한 전이 발생기 - 이 전이 발생기는 제2 매핑된 미디어 객체를 생성하기 위해 상기 수신된 제2 미디어 객체의 복수의 정점을 상기 전이 패턴 메쉬에 매핑함 -, 및

상기 제1 매핑된 미디어 객체 및 상기 제2 매핑된 미디어 객체로부터 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하기 위한 그래픽 프로세서를 포함하는 시스템.

청구항 10.

제9항에 있어서, 상기 그래픽 프로세서는 상기 제1 매핑된 미디어 객체, 상기 제2 매핑된 미디어 객체, 및 알파 맵으로부터 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하며,

상기 알파 맵은 상기 제1 매핑된 미디어 객체와 상기 제2 매핑된 미디어 객체가 교차하는 엣지를 따라 있는 복수의 픽셀에 대응하는 알파 값을 포함하는 시스템.

청구항 11.

제9항에 있어서, 상기 전이 패턴 메쉬는 3차원 입방체이며,

상기 3차원 입방체는 x, y 및 z 좌표 파라미터를 포함하는 시스템.

청구항 12.

제9항에 있어서, 상기 전이 패턴 메쉬의 z 좌표 파라미터는 렌더링 진행의 함수인 시스템.

청구항 13.

제9항에 있어서, 상기 인터페이스는 제3 미디어 객체를 더 수신하며,

상기 그래픽 프로세서는 또한 상기 제1 매핑된 미디어 객체, 상기 제2 매핑된 미디어 객체 및 상기 제3 미디어 객체로부터 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하는 시스템.

청구항 14.

제1 미디어 객체와 제2 미디어 객체 간의 전이를 발생하기 위한 시스템으로서,

상기 제1 미디어 객체 및 상기 제2 미디어 객체를 렌더링하기 위한 제1 프로세서 또는 제1 일련의 명령어,

상기 제1 미디어 객체, 상기 제2 미디어 객체 및 전이 패턴 메쉬를 수신하기 위한 인터페이스,

제1 매핑된 미디어 객체를 생성하기 위해 상기 수신된 제1 미디어 객체의 복수의 정점을 제1 메쉬에 매핑하기 위한 전이 발생기 - 이 전이 발생기는 제2 매핑된 미디어 객체를 생성하기 위해 상기 수신된 제2 미디어 객체의 복수의 정점을 상기 수신된 전이 패턴 메쉬에 매핑함 -, 및

상기 제1 프로세서 또는 상기 제1 일련의 명령어와 독립적으로, 상기 제1 매핑된 미디어 객체 및 상기 제2 매핑된 미디어 객체로부터 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하기 위한 제2 프로세서 또는 제2 일련의 명령어를 포함하는 시스템.

청구항 15.

제14항에 있어서, 상기 제2 프로세서 또는 상기 제2 일련의 명령어는 상기 제1 매핑된 미디어 객체, 상기 제2 매핑된 미디어 객체, 및 알파 맵으로부터 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하며,

상기 알파 맵은 상기 제1 매핑된 미디어 객체와 상기 제2 매핑된 미디어 객체가 교차하는 엣지를 따라 있는 복수의 픽셀에 대응하는 알파 값을 포함하는 시스템.

청구항 16.

제14항에 있어서, 상기 제2 프로세서를 위해 정점 데이터, 텍스처 데이터 또는 스텐실 데이터 중 적어도 하나를 저장하기 위한 프레임 버퍼를 더 포함하는 시스템.

청구항 17.

제14항에 있어서, 상기 전이 패턴 메쉬는 3차원 입방체이고,

상기 3차원 입방체는 x, y 및 z 좌표 파라미터를 포함하는 시스템.

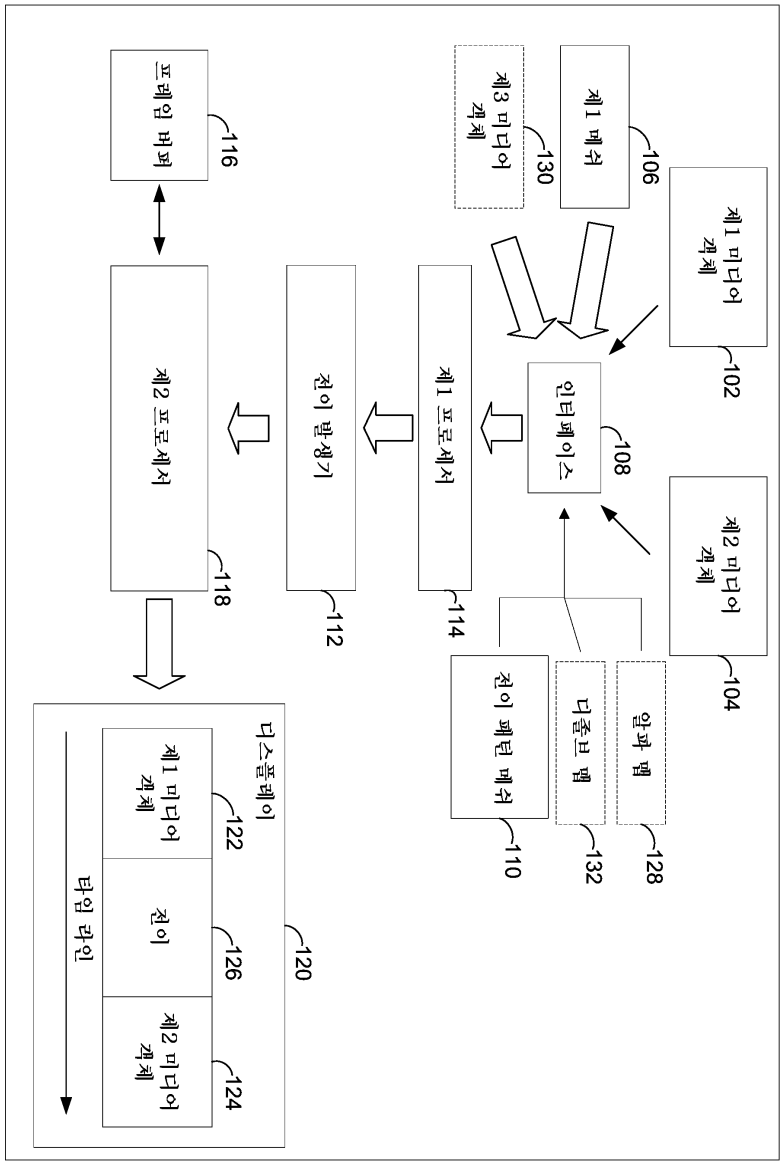
청구항 18.

제14항에 있어서, 상기 인터페이스는 디졸브 맵을 더 수신하며,

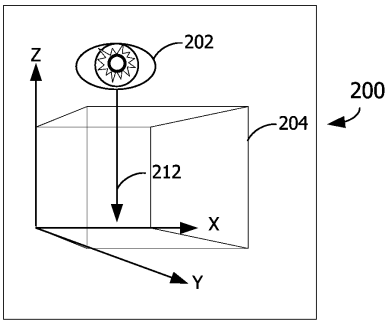
상기 제2 프로세서 또는 상기 제2 일련의 명령어는 또한 상기 제1 매핑된 미디어 객체, 상기 제2 매핑된 미디어 객체 및 상기 디졸브 맵 내의 디졸브된 알파 값으로부터 상기 제1 미디어 객체와 상기 제2 미디어 객체 간의 상기 전이를 렌더링하는 시스템.

도면

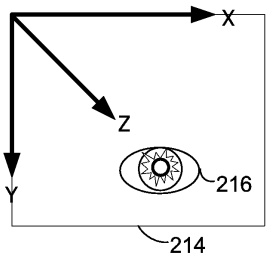
도면1



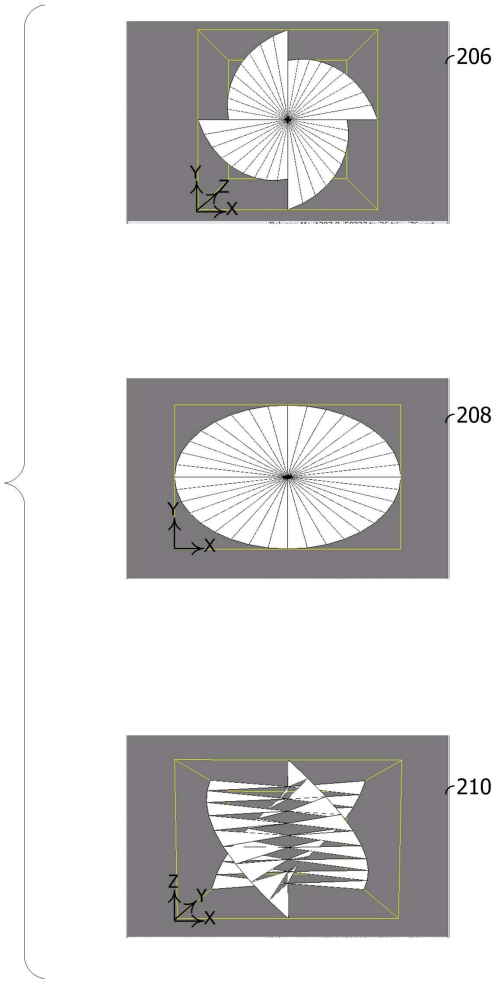
도면2a



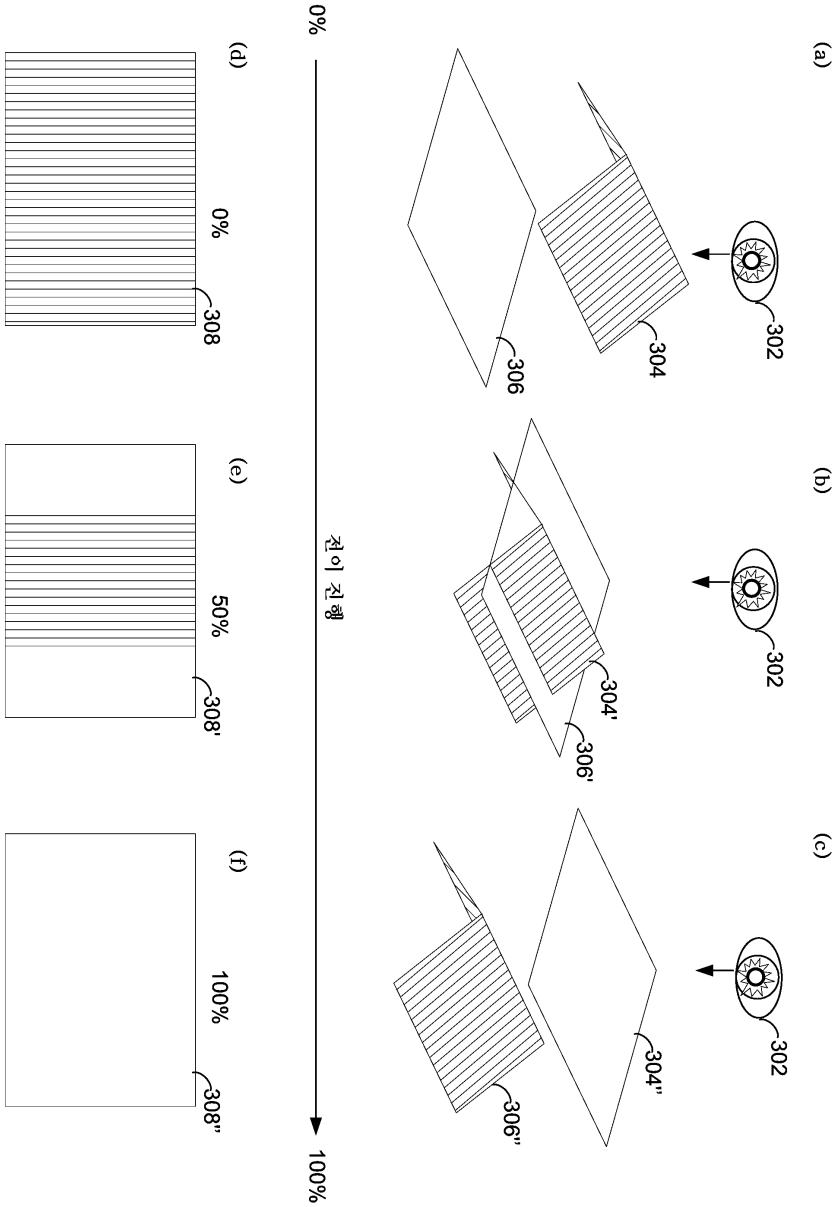
도면2b



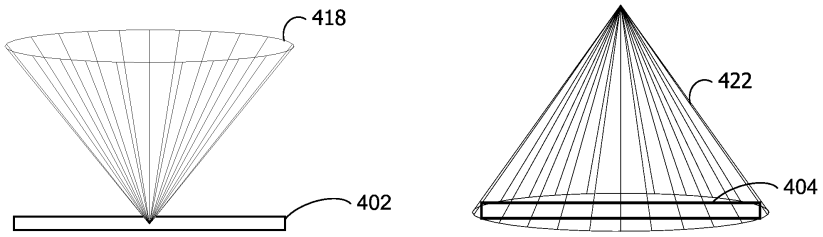
도면2c



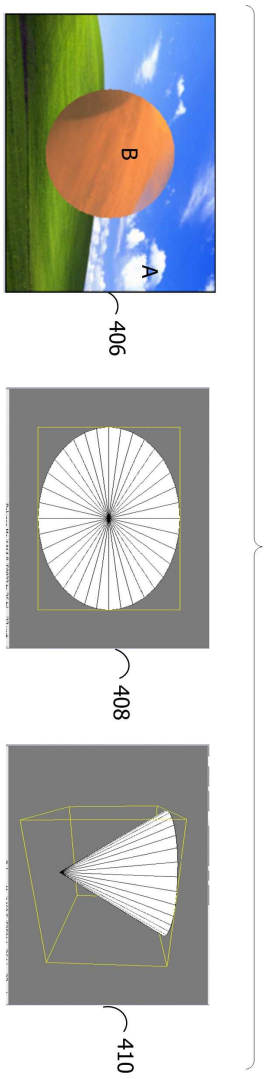
도면3


0%
100%
전이 진행
(a)
(b)
(c)
(d)
(e)
(f)
302
304
306
304'
306'
304"
306"
0%
50%
100%
308
308'
308"

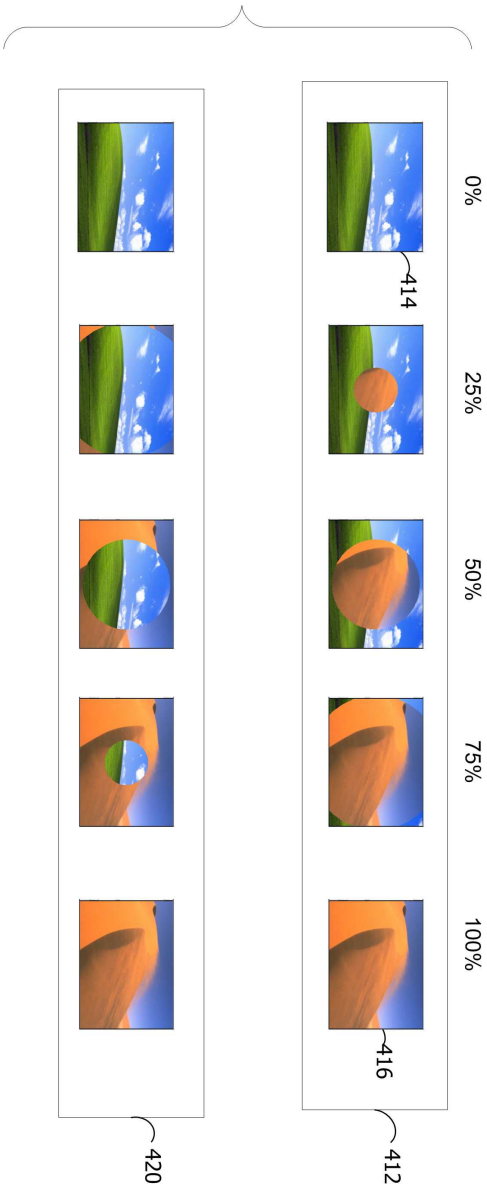
도면4a


418
402
422
404

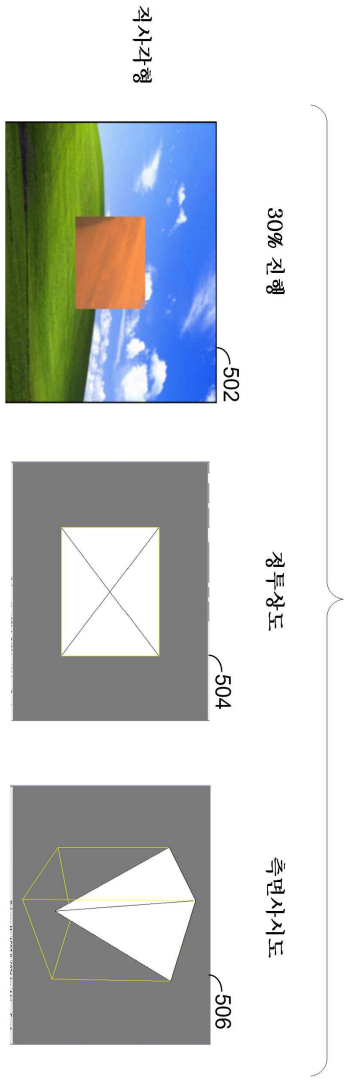
도면4b



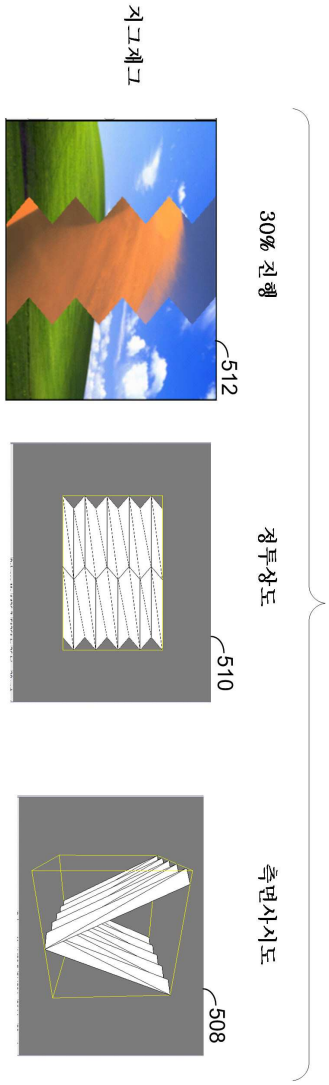
도면4c



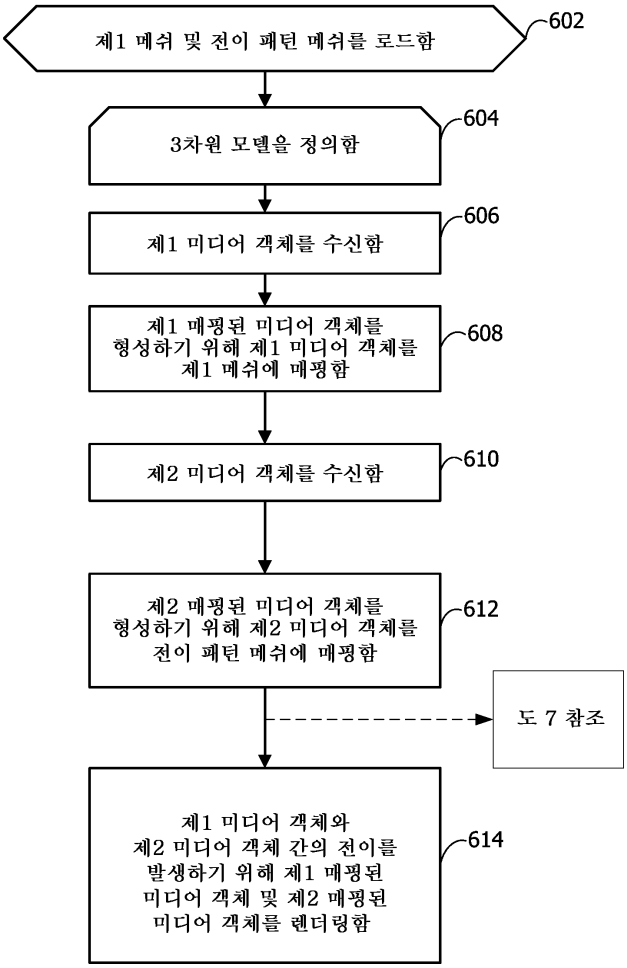
도면5a



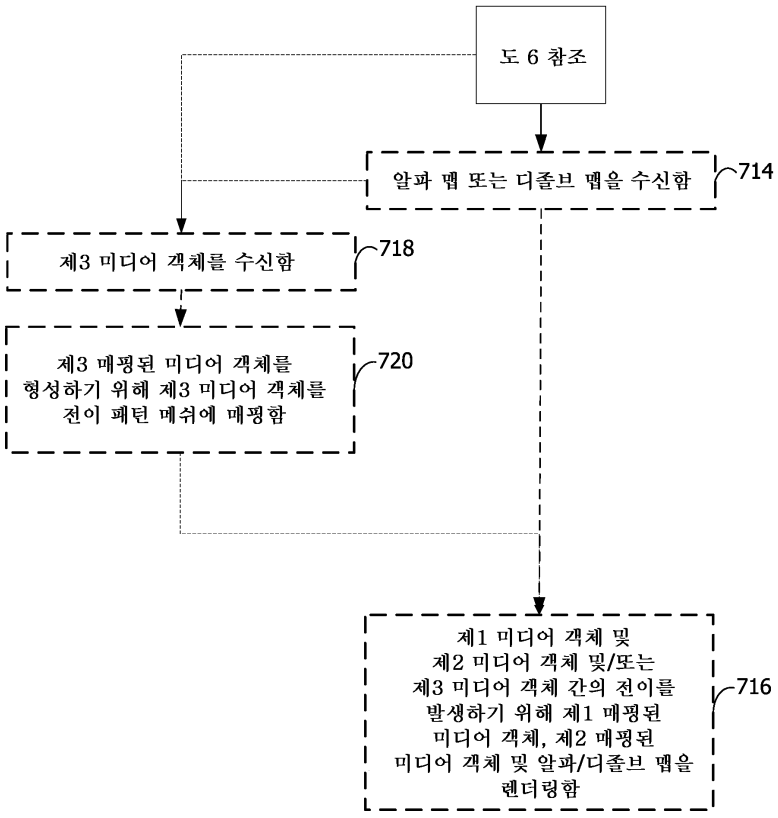
도면5b



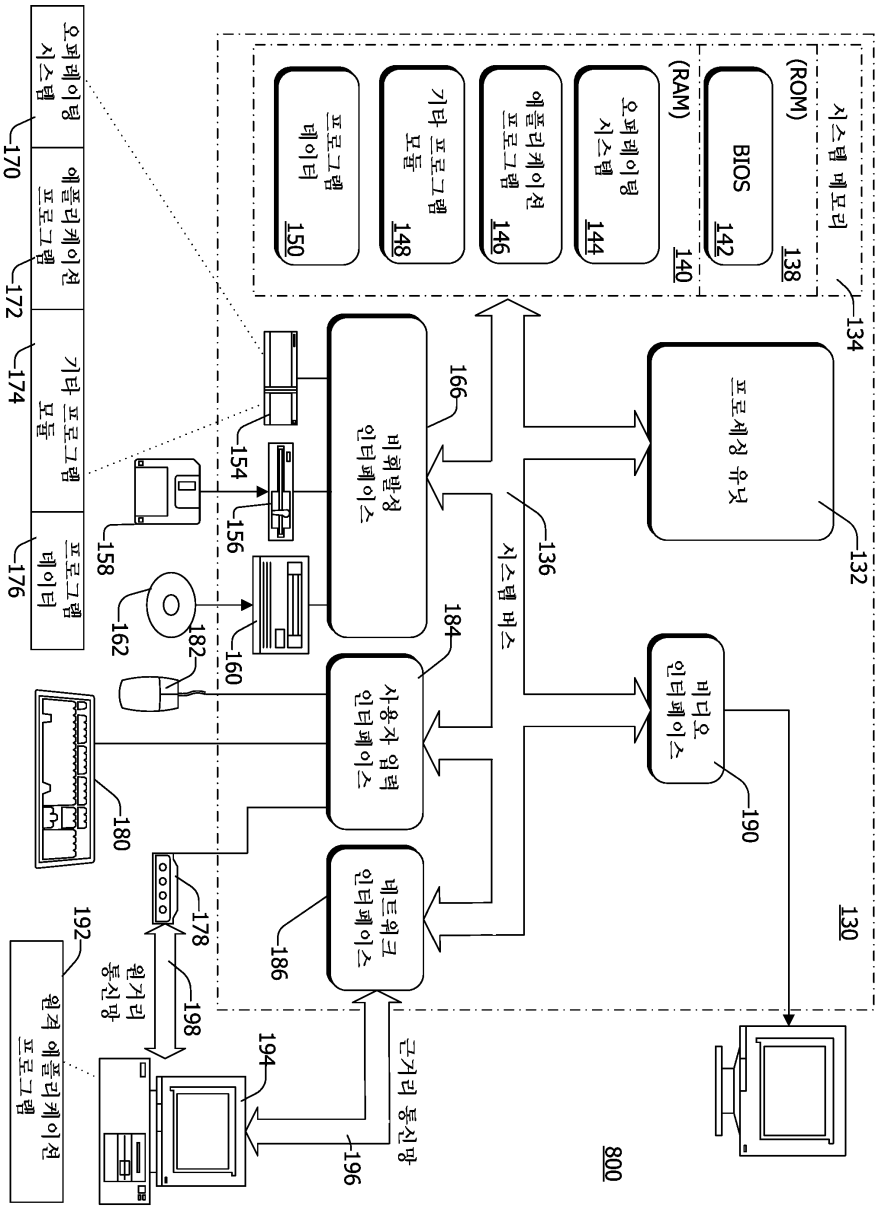
도면6


제1 메쉬 및 전이 패턴 메쉬를 로드함
602
3차원 모델을 정의함
604
제1 미디어 객체를 수신함
606
제1 매핑된 미디어 객체를 형성하기 위해 제1 미디어 객체를 제1 메쉬에 매핑함
608
제2 미디어 객체를 수신함
610
제2 매핑된 미디어 객체를 형성하기 위해 제2 미디어 객체를 전이 패턴 메쉬에 매핑함
612
도 7 참조
제1 미디어 객체와 제2 미디어 객체 간의 전이를 발생하기 위해 제1 매핑된 미디어 객체 및 제2 매핑된 미디어 객체를 렌더링함
614

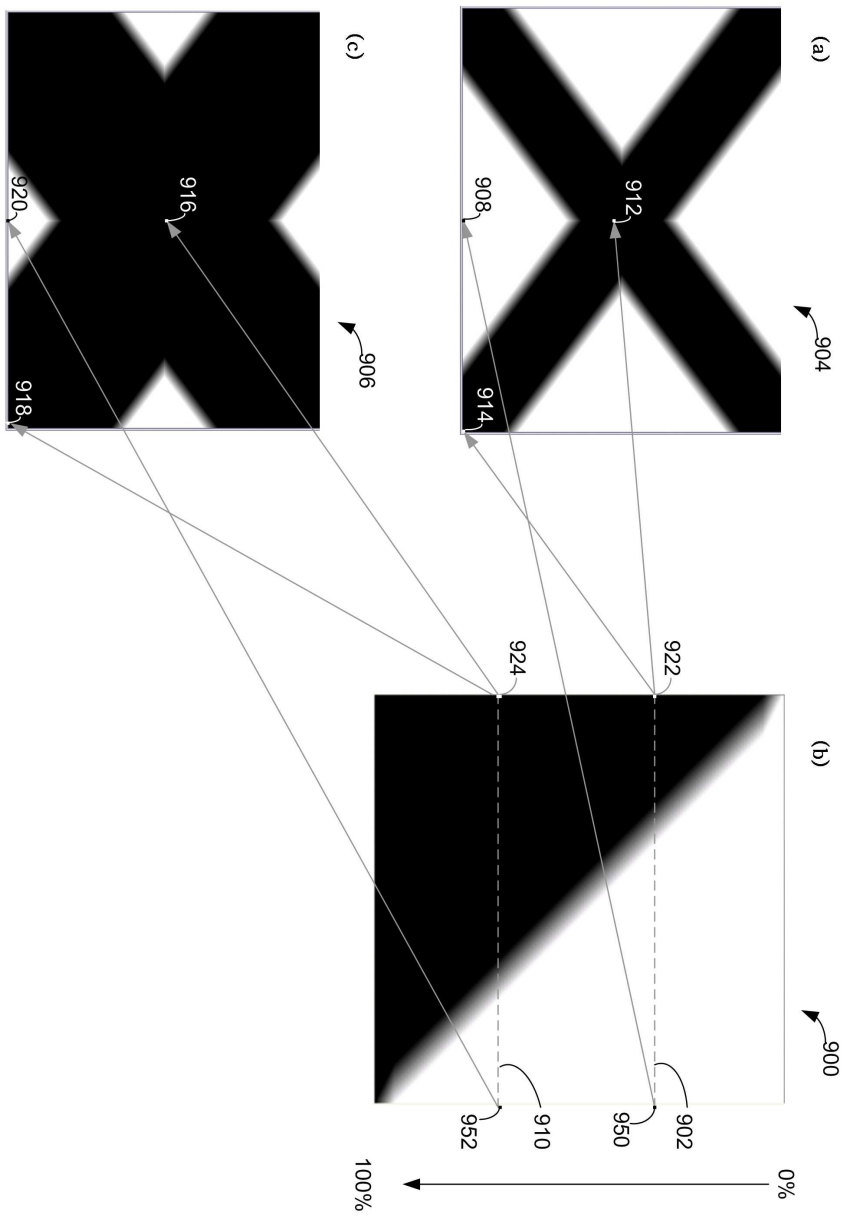
도면7


도 6 참조
알파 맵 또는 디졸브 맵을 수신함
714
제3 미디어 객체를 수신함
718
제3 매핑된 미디어 객체를
형성하기 위해 제3 미디어 객체를
전이 패턴 메쉬에 매핑함
720
제1 미디어 객체 및
제2 미디어 객체 및/또는
제3 미디어 객체 간의 전이를
발생하기 위해 제1 매핑된
미디어 객체, 제2 매핑된
미디어 객체 및 알파/디졸브 맵을
렌더링함
716

도면8


시스템 메모리
(ROM) 138
BIOS 142
(RAM) 140
오퍼레이팅 시스템 144
애플리케이션 프로그램 146
기타 프로그램 모듈 148
프로그램 데이터 150
134
프로세싱 유닛
132
시스템 버스
136
비휘발성 인터페이스
166
비디오 인터페이스
190
사용자 입력 인터페이스
184
네트워크 인터페이스
186
130
800
154
156
158
160
162
180
182
178
192
원거리 통신망
198
194
196
근거리 통신망
원격 애플리케이션 프로그램
오퍼레이팅 시스템
170
애플리케이션 프로그램
172
기타 프로그램 모듈
174
프로그램 데이터
176

도면9


(a)
908
912
914
904
(c)
920
916
918
906
924
922
(b)
900
952
910
950
902
100%
0%

도면9a

