가상현실 시뮬레이션 개발 툴킷 VTree를 이용한 비행 시뮬레이션 구현

조경은, 여인효, 노기석, 이금희, 윤정석, 조형제
동국대학교 컴퓨터/멀티미디어공학과

Implementation of Flight Simulation using VTree SDK

Kyungeun Cho, Inhyo Yeo, Keeseok No, Kumhee Lee, Jungseok Yun, Hyungje Cho
Dept. of Computer Multimedia Engineering, Dongguk University
E-mail : {cke, withgod, myzik, moor0510, poweryun, chohj}@dongguk.edu

요 약

VTree는 개발시간을 감소시키고 실시간 3차원 그래픽스 응용프로그램의 성능을 향상시키는 객체지향 OpenGL기반 소프트웨어 개발 툴킷으로 비주얼 시뮬레이션과 가상현실 응용프로그램을 빠르게 구현하게 해주는 개발 환경을 제공한다.

이 논문에서는 VTree 툴킷에서 제공해주는 다양한 특징들을 소개하고, 이 특징들을 이용하여 구현한 비행 시뮬레이션의 구현과정을 기술한다. 본 연구팀에서 구현한 비행 시뮬레이션의 구현내용은 비행기의 수직 360도 회전비행, 수평 360도 회전비행, 여러 비행기의 교차비행 외에 몇 개의 동작들을 구현하였다. 몇 가지 비행 동작을 구현하는 과정에서 VTree 툴킷을 사용한 방법을 기술하며, 구현과정에서 발생된 몇 가지 문제점들과 향후 연구과제를 소개하는 것이 이 논문의 목적이다.

1. 서론

가상현실이란 컴퓨터를 이용하여 실제환경과 유사하게 만들어진 가상공간 속에서 인간 감각계와의 상호작용을 통해 공간적, 물리적 제약에 의해 현실세계에서는 직접 경험하지 못하는 상황을 간접 체험할 수 있도록 하는 기술이다.

가상 현실 기술은 이미 60년대부터 국방 분야에서 전투기 비행 훈련 분야에서 개발되었고, 학문적으로도 60년대말부터 계속 연구하여 왔다. 그러나 가상현실을 구현하는 기술은 비용이나 기술면에서 연구소나 대학 수준에서 시도하기 어려운 투자가 필요하였다. 가상현실이 대중화되기 시작한 시기는 80년대이다. 80년대 NASA에서 범용 그래픽 워크스테이션에서 우주왕복선의 컴퓨터 모형의 비행시 받는 공기 저항 패턴을 해석하고 그 결과를 가시화 하는데 가상 현실 기술을 적용하였다. 이는 저가형 그래픽 워크스테이션에서 가상현실 기술을 응용분야에 적용하는 가능성을 제시하였고, 이를 계기로 일반 기업이나 연구소에서 다양한 분야에 가상 현실 기술을 적용하기 시작하였다.

가상 현실을 구현하는 요소에는 세계 모델링, 실시간 그래픽 렌더링, 사용자 상호작용, 실시간 시뮬레이션, 다양한 감감 표시기 구동 기술이 필요하다. 가상현실을 응용하는 산업분야는 다양하다. 토목, 건축, 인테리어, 기계 분야에서와 같이 실제의 공간 또는 물품을 만들어야 경험해 볼 수 있는 분야에 가상현실이 대폭 운영되고 있으며 특히 자본/시간/위험성과 관련된 분야에 가상현실은 가장 유용한 효용성을 지니고 있다. 또한 원자력 발전소처럼 위험성이 있어 실제 공간에 들어가서 작업 할 수 없는 분야, 병원, 항공기및 전차훈련과 같이 실제로 연습해 보기 어려운 분야, 과학, 자연현상, 우주탐험처럼 눈으로 볼 수 없는 것을 가시화하는 분야, 그리고 게임이나 Edutainment과 같은 교육 및 오락 분야가 있다[1].

가상현실을 이용한 시뮬레이터는 군사용과 민간용

으로 구분할 수 있으며 특히 군사용 목적에서 엄청난 비용으로 제작되는 장비에 대한 사전 점검 및 기능을 시험할 수 있는 방법을 제공한다. 주로 탱크, 전투기, 헬리콥터, 미사일, 개인 화기 등이 가상현실 시뮬레이션 및 훈련용으로 활용되고 있다. 민간 항공 시뮬레이터는 세계적인 항공사들이 자체 비행사 훈련 프로그램의 일환으로 가상현실을 이용한 항공기 조정 시뮬레이터를 도입하여 활용하고 있다. 이는 항공기를 직접 조정하기 전에 시뮬레이션 환경에서 모의 조정함으로 비상 상태를 파악하고 조치할 수 있도록 하는 것이다. 비행 시뮬레이터는 실제로 항공기에 탑승하여 비행하지 않으면서도 마치 항공기에 탑승한 듯한, 즉 현실감 있는 비행 체험을 사용자로 하여금 경험하게 하기 때문에 실제 비행에 드는 엄청난 인적, 물적 자원을 절약할 수 있고, 또한 알려지지 않은 항로나 접근이 금지된 지역의 비행을 시뮬레이션할 수 있기 때문에 실제 비행에서 존재할 수 있는 위험을 예방할 수 있다.

비행 시뮬레이션 개발의 어려움으로는 실제 비행기의 정확한 데이타와 고급 그래픽 프로그래밍을 들 수가 있다. 3차원 그래픽의 구현에 필수적인 기법인 폴리곤 기법과 텍스쳐 매핑, 레이트레이싱 등의 까다로운 분야가 많다. 실제 그래픽 프로그래밍은 많은 수학적인 이해와 기반을 가지고 있어야 한다. 이러한 이유가 시뮬레이션 프로그램을 더욱 어렵게 하는 요소가 될 수 있다[2,3].

VTree 툴킷은 쉽게 비행 시뮬레이션과 같은 응용프로그램을 제작할 수 있게 지원을 한다. 본 연구에서는 일반적인 시뮬레이션 프로그래밍의 어려운 요소들을 극복하게 해주는 VTree 툴킷을 이용하여 비행 시뮬레이션을 구현한 방법을 소개한다. 비행 시뮬레이션의 구현내용은 비행기의 수직 360도 회전비행, 수평 360도 회전비행, 여러 비행기의 교차비행 외에 몇 개의 동작들을 구현하였다. 몇 가지 비행 동작의 구현과정과 VTree 툴킷을 사용한 방법을 기술한다.

본 논문의 구성은 다음과 같다. 2장에서는 VTree 툴킷에서 제공해주는 다양한 기능과 특징들을 소개하고, 3장에서는 이 특징들을 이용하여 비행시뮬레이션을 구현한 과정을 기술한다. 4장에는 구현결과를 제시하며 마지막에 향후 연구 과제를 언급하고 결론을 맺는다.

2. 가상현실 시뮬레이션 개발 툴킷 VTree의 기능 소개

VTree는 개발시간을 감소시키고 실시간 3차원 그래픽스 응용프로그램의 성능을 향상시키는 객체지향 OpenGL기반 소프트웨어 개발 툴킷으로 비주얼 시뮬레이션과 가상현실 응용프로그램을 빠르게 구현하게 해주는 개발 환경을 제공한다. VTree 툴킷의 몇 가지 특징들은 다음과 같다.

(1) 큰 지형 페이지를 실시간으로 쉽게 시각화하는 기능: 전에는 실시간에서는 실행되기에 불가능이라 여겨졌던 매우 큰 규모의 지형을 VTree에서 제공하는 함수로 쉽게 장면에 추가할 수 있다. VTree는 자동적으로 큰 규모의 지형을 페이징을 하는 기능을 내부적으로 수행한다.

(2) 소리, 특수효과, 충돌처리를 위한 다양한 라이브러리 지원: 특수효과로는 섬광, 폭발, 파편, 화염, 연기, 먼지, 미사일이나 항공기 등에서 분출되는 연기를 지원하고 3차원 음향 효과 등을 라이브러리로 지원한다. 또한 비행석의 계기판 등을 구성하기 위해서 많은 시간을 투자할 필요없이 라이브러리화되어 있는 기능을 쉽게 구현가능하다. 적외선 기능이나 야간 투시경을 통하여 물체를 보는 기능도 지원한다.

(3) 간소화된 OpenGL 프로그래밍: 고속 3차원 그래픽 처리를 제공하기 위하여 업계표준에 근거하는 OpenGL 기능을 완전히 이용하며, 폴리곤이나 라인 등의 프리미티브 그래픽 프로그래밍 대신에 이미 모델링된 오브젝트와 프로그램과의 상호작용만을 프로그래밍하면 된다.

(4) 업체 표준 파일 포맷 지원: 기존의 3차원 모델 데이타를 VTree파일 포맷으로 변형하는 툴을 제공하여 Digital Elevation Terrain Data (DTED), Wavefront (OBJ), Coryphaeus (DWB), MultiGen OpenFlight (FLT) 등과 같은 파일을 모두 사용가능하다.

(5) 장면 그래프 편집기 제공: 하나의 VTree 파일의 노드를 편집, 삭제, 또는 추가할 수 있다. 다중 LOD(Level of Details), articulations, animations를 툴을 이용하여 편집가능하다.

(6) 충돌 감지 영역 편집기: 손으로 오브젝트의 바운딩 박스를 설정할 수 있는 기능을 제공하여 가장 적합한 충돌 감지 영역을 설정할 수 있다. 개발자들은 응용프로그램에서 가장 적합한 충돌감지 경계를 위한 정확한 수준을 조정할 수 있다. VTree의 충돌 검지 라이브러리는 프로그래머가 물체가 충돌할 때 결정하기 위한 메카니즘을 제공한다.

(7) 속도 최적화된 풍부한 C++ 라이브러리 제공: 효과적인 뷰포트 설정 기능이 제공되며, 마우스, 조이스

틱, 헤드 트랙커, 스테레오 안경, HMD과 같은 가상 현실 주변 장치 인터페이스를 제공한다. LOD(Level Of Details) 스위칭 기능, 충돌 검지, 교차점 테스팅, 편리한 이벤트 핸들러 기능을 라이브러리에서 제공한다.

앞에서 제시한 VTree 툴킷의 다양한 기능을 이용하면 짧은 시간내에 쉽게 가상 현실 시뮬레이션 시스템 구현이 가능하나 단점은 가격이 매우 고가라는 점이다[4,5,6].

3. 비행 시뮬레이션의 구현

본 과제에서 구현한 비행 시뮬레이션은 비행기의 수직 360도 회전비행, 수평 360도 회전비행, 여러 비행기의 교차비행 외에 몇 개의 동작들을 구현하였으며, VTree에서 제공하는 기능들을 이용하여 사운드와 특수효과 처리 기능을 구현하였다. 다음 각 단계에서는 VTree에서 제공하는 라이브러리를 사용하여 구현한 방법들을 기술한다.

3.1 VTree에서의 좌표설정 기준 및 뷰포트 설정

VTree는 윈도우즈 설정을 위해 OpenGL 렌더링을 위해 초기화된 "hWnd" 인수를 받아들이기 때문에 응용프로그램에서 vtWindow 오브젝트를 위한 Win32 윈도우즈를 생성할 수 없다. 즉, 그림1과 같은 좌표가 설정된다.

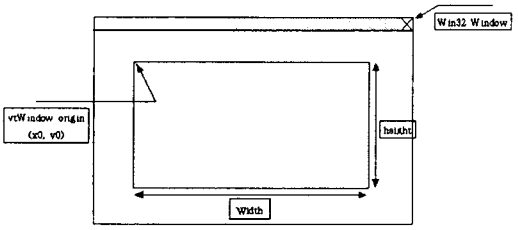

그림 1. vtWindow 좌표체계

비행기의 비행을 모의실험하는 프로그램은 관찰자가 3차원 상의 원하는 곳에 존재할 수 있도록 하는 것이다. 뷰시스템에서 관찰자의 현재 좌표를 지나칠 때 뷰시스템은 3차원 세계에서 그 좌표를 지날 때 보여질 수 있는 것으르 뷰포트에 보여준다. 윈도우즈와 뷰포트의 적용 단계가 다음과 같다. CreateViewport 함수를 사용하여 뷰포트를 생성하고, 모든 뷰포트에 장면(scene)을 삽입한다. 이때 모든 뷰포트는 렌더링을 위하여 장면을 소유해야 한다. 그 다음 단계는 각각의 뷰포트에 대한 투영 방식을 설정하고 렌더링을 하는데, 이때 렌더링은 vtWindow 오브젝트에 의해서가 아니라 뷰포트에 의해서 실행된다(소스 1).

```
mainView = window->CreateViewport("Main View",0,0,900,900);
vtScene *scene;
mainView->SetScene(scene); // viewport scene을 "scene"에 설정
mainView->SetPerspective(vtDToR(45.0f), 1.0f, 1.5f, 100.0f)
        또는
mainView->SetOrtho(-1.0f, 1.0f, -2.0f, 2.0f, -5.0f, 5.0f)
window->Render();
```

소스 1. 뷰포트 설정

VTree는 다중 뷰포트(Multiple Viewport)를 지원한다. 그림 2에 해당하는 다중 뷰포트를 설정하는 과정은 소스 2와 같다. 각각의 뷰포트에 대하여 모든 설정을 정의한다. 각 뷰포트이 대한 설정을 해주는 것만으로 동시에 다수의 화면에 대한 카메라의 동작 및 뷰포트에 대한 투영 방식을 별도로 제어할 수 있다.

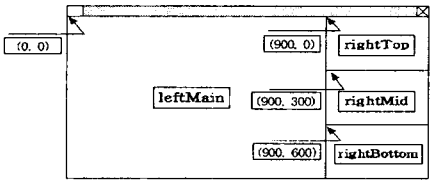

그림 2 다중뷰 설정

```
win = new vtWindow("Window",vtWinhWnd,0,0,1200,900);
        // windows 전체영역 설정
leftMainView = win->CreateViewport("leftMain View",0,0,900,900);
rightTopView = win->CreateViewport("rightTop View",900,0,300,300);
rightMidView = win->CreateViewport("rightMid View",900,300,300,300);
rightBottomView
        = win->CreateViewport('rightBottom View",900,600,300,300);
```

소스 2 . 다중뷰 설정

3.2 비행기의 회전 동작 구현

비행기의 회전하는 부분은 2차원 그래픽에서 사용되는 점의 회전 공식을 사용한다. 이 프로그램에서 각기 동작을 나누는 구분은 시간을 사용한다. 일정한 시간동안 직진을 하는 비행기가 원하는 시간이 되면 다음 동작을 넘어가게 되는데 다음 동작이 처음 시작되는 좌표값을 기억한다. 여기서 기억된 좌표값은 차후에 한바퀴를 돌고 다시 그 지점으로 돌아오는 지를 검사하기 위해 사용된다. 시뮬레이션은 3차원 공간상에서 이루어지지만 비행기의 회전에서는 두 개의 축에 대해서만 운동하므로 2차원 공식을 사용한다.

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

처음 시작되는 좌표를 저장했으면 점의 회전 공식을 사용하는데 이 공식은 원점을 중심으로 θ만큼 점을 회전시키면 된다. 그러나 비행기는 원점 가까이 있지 않고 또한 원점을 중심으로 회전하는 것이 아니므로 약간의 수정이 필요하다. 원하는 중점을 원점으로 보냈다가 θ만큼 회전시키고 다시 원래의 점으로 되돌아오는 방법을 사용한다. 이것은 현재의 좌표 값들을 빼주었다가 회전을 시킨 후에 다시 더하는 방법을 사용하는 것이다. 이렇게 비행기를 회전시킬 때 그 비행기의 몸체도 회전을 해서 실제 비행기가 움직이는 것처럼 보여야 하나 비행기의 몸체는 비행기가 움직인 y축, x축의 이동 거리만큼 atan2 값을 이용해서 회전시키게 된다(소스 3).

```
roll_x_x = ((cos((ang))*rotation_x) - (sin((ang))*rotation_y))+roll_x;
roll_y_y = ((sin((ang))*rotation_x) + (cos((ang))*rotation_y))+roll_y;
vtFloat heading = atan2(-(to.y - vehiclePos.y),(to.x - vehiclePos.x));
```

소스 3. 회전 동작

마지막으로 비행기가 한바퀴 회전을 해서 초기 위치로 돌아왔는지를 비교하기 위해서 초기에 저장했던 값에서 y축 값(여기서는 고도가 됨)만을 비교하여 같으면 다음 단계로 넘어가게 된다.

3.3 교차 비행의 구현

Vtree를 이용한 비행기 4대의 교차비행은 다음과 같은 방법으로 구현한다.

```
Model_2_InitialPos.x = TerCenter.x-50;
Model_2_InitialPos.y = TerCenter.y-46;
Model_2_InitialPos.z = TerCenter.z+100;
vtSet(mat2, Model_2_InitialPos)

Model_3_InitialPos.x = TerCenter.x-250;
Model_3_InitialPos.y = TerCenter.y-52;
Model_3_InitialPos.z = TerCenter.z+100;
vtSet(mat3, Model_3_InitialPos)

Model_4_InitialPos.x = TerCenter.x-50;
Model_4_InitialPos.y = TerCenter.y-98;
Model_4_InitialPos.z = TerCenter.z-100;
vtSet(mat4, Model_4_InitialPos)

Model_5_InitialPos.x = TerCenter.x-250;
Model_5_InitialPos.y = TerCenter.y-102;
Model_5_InitialPos.z = TerCenter.z-100;
vtSet(mat5, Model_5_InitialPos)
```

소스 4. 비행기의 초기 좌표 설정

TerCenter는 지형의 정중앙 좌표이다. 구현할 때 x좌표값은 그림 3의 Model_2와 Model_4가 같게 하고 Model_3과 Model_5가 같게 한다. 또, Model_2와 Model_3의 x좌표값의 차와 Model_4과 Model_5의 x좌표값의 차를 같게 한다. y좌표값은 서로 충돌하지 않을 정도의 높이 차만 준다. z좌표값은 Model_2과 Model_3가 같고 Model_4과 Model_5가 같게 한다. 교차비행을 위한 초기 좌표 설정은 완료되었고 비행기는 90도 간격으로 교차비행할 준비가 되었다(소스 4).

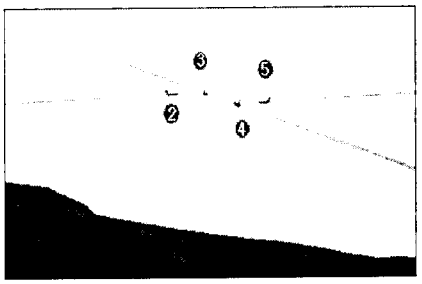

그림 3. 비행기 4대의 교차비행 모델

그러나, 앞에서처럼 구현할 경우 비행기는 정확히 교차 비행을 하지 않고 원하는 결과와는 조금 혹은 많이 어긋나게 비행을 하게 된다. 그 이유는 비행기의 Heading을 제대로 안 잡아 주었기 때문이다. 이 문제는 비행기가 모두 한 방향을 바라 보기 때문이다.

```
vtRotateY(mat2, vtDToR(90));
vtRotateY(mat3, vtDToR(45));
vtRotateY(mat4, vtDToR(225));
vtRotateY(mat5, vtDToR(0));
```

소스 5. 비행기의 Heading 설정

따라서 이 문제를 해결하기 위하여 각 비행기에 맞는 로테이션 값을 주어 목표 지점을 바라보게 세팅해주면 문제가 해결된다. 소스 5 부분에서는 vtRotateY라는 함수를 사용하여 설정해준다. 여기서 각도는 라디안값이다. vtRotateY는 현재 Y축을 기준으로 회전시키는 함수이다. 따라서 교차비행 Heading 설정하는 부분은 한번만 호출한다. 그렇지 않으면 Model_2 비행기는 90도 만큼씩, Model_3은 계속 45도 만큼씩 빙빙 돌게 될 것이기 때문이다. 소스 6처럼 vtSet 함수를 써서 이동하는 부분을 구현한다. 앞에서 저장된 좌표를 월드에 뿌려주면 교차 비행이 수행된다(소스 7).

```
vtSet(pos,(Model_2_InitialPos.x+250)-(count*
40),Model_2_InitialPos.y-(count*2),
(Model_2_InitialPos.z+250)-(count*40));

// 거리를 더 늘리고 싶으면
// Model_2_InitialPos.x 는 + (num),
// Model_2_InitialPos.z 는 +(num)

vtSet(pos2,(Model_3_InitialPos.x-250)+(count
*40),Model_3_InitialPos.y-(count*2),
(Model_3_InitialPos.z+250)-(count*40));

// 거리를 더 늘리고 싶으면
// Model_3_InitialPos.x 는 - (num),
// Model_3_InitialPos.z 는 +(num)

vtSet(pos,(Model_2_InitialPos.x+250)-(count*
40),Model_2_InitialPos.y-(count*2),
(Model_2_InitialPos.z+250)-(count*40));
```

소스 6. 비행기의 실제 이동

```
Model2->SetModelToWorld(mat2);
Model3->SetModelToWorld(mat3);
Model4->SetModelToWorld(mat4);
Model5->SetModelToWorld(mat5);
```

소스 7. 월드에 나타내기

3.4 사운드와 특수 효과 처리

Vtree에서 사운드처리는 간단하게 플레이할 수 있도록 라이브러리를 지원한다. 사운드는 vtSoundFX 함수를 이용하는데 사운드파일을 읽어들인 후 SoundFX.Action에서 StartEngine 함수로서 원하는 파일을 플레이 한다. 여기서 SoundFX.Action에서 파라미터를 다르게 줌으로써 다양한 효과를 낼 수 있으며, 사운드를 좌,우를 조절하는 기능이 있다. 실제 사용 예는 다음과 같다(소스 8).

```
vtSounFX SoundFX;
SoundFX.BufferLength(0.15);
SoundFX.hWnd(vtWinhWnd);
Sound1 = SoundFX.AddSound("sound1.wav");
SoundFX.Action(vtSoundFX::StartEngine);
SoundId =
   SoundFX.Action(vtSoundFX::Play,(vtUint32)-1,0,Sound1,-1,1.0,0.7);
SoundFX.Action(vtSoundFX::StopEngine);
```

소스 8. 사운드 효과 설정

Vtree를 이용한 특수효과 중에 미사일 효과를 사용한다. 미사일이 날아갈 때 비행기의 뒷부분에 연기가 나오는 특수효과이다. 특수효과는 다음과 같이 구현한다(소스 9). SetTexture라는 함수를 이용하여 TrSmoke3.rgba란 텍스쳐를 사용한 것을 볼 수 있다. for 문은 표현하고자 하는 비행기수 만큼 반복문을 돌려준 것이다.

```
vtFxMissileTrail *mslTrail[8] = {NULL};

for(i=0; i<8; i++){
mslTrail[i] = new vtFxMissileTrail("My Trail
Effect 1", trailPos, trailDuration, trailSize);

mslTrail[i]->SetTexture("TrSmoke3.rgba");
mslTrail[i]->SetTexScale(0.05f, 1.0f);
mslTrail[i]->SetColor(vtColor(0.8f, 0.8f, 0.8f,
1.0f));
mslTrail[i]->SetFadeDuration(trailDuration);
mslTrail[i]->SetRotRate(vtDToR(180.0f));
mslTrail[i]->SetAddInterval(0.1f);
mslTrail[i]->SetExpandTime(8.0f);
}
```

소스 9. vtFxMissileTrail 특수효과 설정

```
Model1->AddFx(mslTrail[0]);
Model2->AddFx(mslTrail[1]);
Model3->AddFx(mslTrail[2]);
Model4->AddFx(mslTrail[3]);
Model5->AddFx(mslTrail[4]);
Model6->AddFx(mslTrail[5]);
Model7->AddFx(mslTrail[6]);
Model8->AddFx(mslTrail[7]);
```

소스 10. vtFxMissileTrail 효과 설정

소스 10에서 보듯이 vtEntity인 Model 1부터 8까지 AddFx란 함수로 특수효과를 첨가한다. 실제로 effect를 적용해주기 위해 소스 11-12와 같이 함수를 구현한다.

```
void effect()
{
vtFloat vel,az,el;

vtFx::RandVal(vel,30.0f, 60.0f);
vtFx::RandVal(az,vtDToR(0.0f),
vtDToR(360.0f));
vtFx::RandVal(el,vtDToR(50.0f),
vtDToR(80.0f));
vtGMatrix  mat_mis;

vtZYXRotation(mat_mis,0.0f,az,el);
vtMultiply(mslVel,mat_mis,vMinusZ);
vtScale(mslVel,vel);
mslLaunchTime = vtFloat(vtTime());
```

소스 11. effect 함수 구현

```
for (int i=0; i<8; i++){
mslTrail[i]->SetGenerate(TRUE);
mslTrail[i]->Resume();
mslTrail[0]->Stop();
mslTrail[1]->Stop();
mslTrail[2]->Stop();
mslTrail[3]->Stop();
mslTrail[4]->Stop();
mslTrail[5]->Start();
mslTrail[6]->Start();
mslTrail[7]->Start();
vtFloat tFlight = 10.0f*mslVel.y/9.8f;
mslTrail[i]->SetDuration(1.2f*tFlight);
mslTrail[i]->SetFadeDuration(1.0f*tFlight);
mslInFlight = TRUE;
    }
}
```

소스 12. effect 함수 구현 (계속)

여기서보면 Model_1에서 5에해당하는mslTrail[0]에서 [4]까지 모두 Stop으로 설정되어 있는 것을 볼 수 있다. 사용되지 않는 비행기의 특수효과를 정지한 것이다.

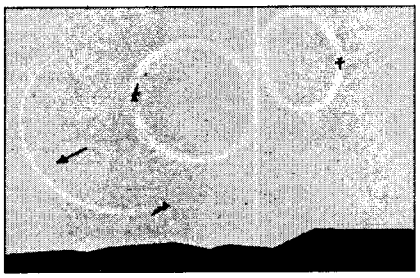

그림 4. 특수 효과 삽입 결과

그림 4의 화살표 부분을 보면 특수효과가 다른 비행기와는 달리 특정 영역에서만 작동하는 것을 볼 수 있다. 그림 4와 같이 구현하는 것은 비교적 간단하다. 만약, 그림 4의 첫 번째 비행기가 mslTrail[0] 이였다면 특수효과를 보여줄 구간이 아닌 경우 mslTrail[0]->Pause();로 일시 정지시키고, 다시 특수효과를 보여주어야 할 영역이 되면 mslTrail[0]->Start();를 시켜주면 된다. 하지만 일시정지로는 완전히 만족스러운 결과를 얻을 수는 없다. 더 완벽한 제어를 원한다면 특수효과를 표현할 구간을 지나면 서서히 특수효과 표현량을 줄여주어 거의 보이지 않도록 구현하면 된다.

4. 구현 결과

비행기 한 대가 x축 방향으로 진행하다 카메라의 중심에 도달하면 수직으로 360도 동작을 수행후 초기 진행 방향을 향하여 빠져나가는 결과가 그림 5이다.

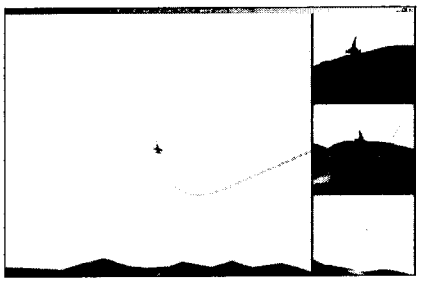

그림 5. 수직 360도 회전 동작 결과

4 대의 비행기가 중앙을 향해 이동하면서 교차비행을 하며 각각 반대 방향으로 빠져 나간 것을 구현한 결과가 그림 6이다.

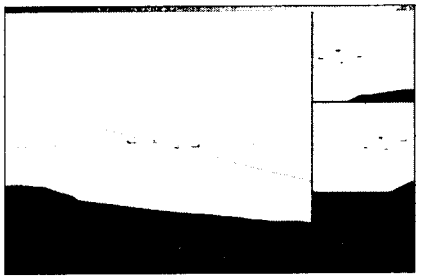

그림 6. 4 대의 비행기 교차 동작 결과

5. 결론 및 향후 연구 과제

VTree 툴킷에서 제공하는 다양한 특징들을 이용하여 구현한 비행 시뮬레이션의 구현과정을 소개하였다. 향후 연구과제로는 VTree에서 제공하는 많은 기능들을 이용하여 좀 더 다양한 비행 동작들을 추가적으로 구현하는 것이며, 이것을 이용하여 시뮬레이션 게임에도 적용하는 방법에 관한 연구가 이루어져야 한다.

참고문헌

[1] 고희동, "Virtual Reality", 한국 CAD/CAM학회지 제3권 제 2호, pp.70-72, 1997

[2] 류승택 외 3인, "3차원 지형 모델링 및 비행 시뮬레이션 구현", 1997년도 한국정보과학회 봄 학술발표논문집, Vol. 24 No.1, pp.387-390, 1997

[3] 정이중, "3차원 시뮬레이션", 에프원, 1998

[4] www.cg2.com/products/vtree/vtreemain.htm

[5] VTree 3.0 Users Guide, 2000

[6] VTree 3.0 Reference Manual, 2000