파티클 시스템을 이용한 게임 및 가상현실에서의 특수효과

송승헌* · 김응곤**

1. 서 론

첨단 디지털 영상 제작기술은 1980년대 이후 헐리웃 영화를 중심으로 컴퓨터를 이용한 특수효과가 많이 사용되면서 큰 발전을 이루게 되었다. 1990년대에 이르러서는 CG 애니메이션과 3차원 게임의 등장으로 본격적인 성장을 맞이하게 되었으며, 텍스쳐나 기타 속성값의 설정에 의해 불이나 연기, 폭발, 액체의 분무, 눈발, 연기 흔적 등의 특수효과를 간단히 구현할 수 있었다.

최근 게임이나 가상현실 등의 분야에서는 파티클 시스템을 이용하여 대부분의 특수효과들을 만들어 내는데, 이러한 파티클 시스템은 여러 개의 개별적인 입자들의 집합을 의미하며, 컴퓨터 그래픽스에서는 입자들을 사용한 모든 기법들을 지칭한다[1,2].

파티클 시스템에서 각각의 입자는 개별적인 특성을 갖고 있으며 입자들은 서로 간에 독립적으로 행동하지만, 주어진 파티클 시스템에서의 입자들은 공통적인 특성을 공유하고 있기 때문에 개별 입자들은 독립적으로 움직이더라도 전체적으로 볼 때 하나의 공통된 효과를 나타낸다. 파티클 시스템을 이용하면 불꽃, 폭발, 연기, 액체, 눈, 비, 먼지와 같은 자연현상을 효과적으로 표현할 수 있으며, CG 애니메이션이나 게임에서도 사실적인 자연현상의 표현을 할 수 있기 때문에 파티클 시스템이 많이 이용되고 있다[3].

특수효과 발생을 위한 상위 수준의 그래픽스 API인 파티클 시스템 API를 개발하게 되면 픽셀 단위가 아닌 객체단위로 특수효과 제작 프로그래밍 작업이 가능하므로 프로그래머가 구현 알고리즘을 모르더라도 단지 그래픽스의 기본 개념만 이해하면 되고, 어떠한 플랫폼에서도 쉽게 포팅할 수 있게 된다[5].

따라서, 본 논문에서는 게임 프로그램이나 가상현실 응용 프로그램에서 실시간으로 동적 파티클 시스템을 이용하여 불, 불꽃, 유체 속성을 갖거나, 기상현상 재현 등과 같은 특수효과를 만들어 낼 수 있도록 하기 위해 파티클 시스템 API를 개발하고자 한다. 이러한 파티클 시스템을 이용하여 구축된 특수효과 라이브러리는 여러 가지 특수효과를 필요로 하는 온라인 게임이나 가상현실 응용프로그램에서 매우 유용하게 사용될 수 있다.

1장 서론에 이어 2장 관련연구에서는 상용 패키지나 다른 그래픽스 라이브러리에서 사용되는 파티클 시스템에 대해 기술한 후, 3장에서 구현하고자하는 파티클 시스템에 대해 정의 및 설계하고, 4장에서는 파티클 시스템을 이용한 특수효과를 구현한다. 5장에서 결론을 내리며 향후 연구과제를 제시한다.

* 순천대학교 대학원 컴퓨터과학과
** 순천대학교 컴퓨터과학과

2. 관련연구

Maya나 3D Studio Max와 같은 상용 패키지에서 특수효과를 만들기 위한 파티클 시스템을 지원하고 있으나, 온라인상에서는 바로 사용이 불가능하며 대화식으로만 제작할 수 있다.[5] 따라서 게임 등의 프로그램에서 특수효과를 내기 위해서는 오프라인 상에서 제작한 특수효과를 사용하거나 자체적으로 특수효과를 내기 위한 기능을 삽입해야 하는데, 이를 위해서는 개발기간이나 비용이 많이 소요된다는 단점을 가지고 있다[6].

그리고 특수효과를 생성하기 위한 기존의 연구들에서도 유체 속성을 갖는 물질들인 불, 불꽃, 폭발 속성을 갖는 물질의 특수효과나 기상현상의 재현을 위한 특수효과 생성 기법을 제시하고 있으나, 모두 개별적인 알고리즘의 생성과 적용으로 특별한 경우의 효과를 표현하기 위한 것이기 때문에 프로그래머가 게임이나 가상현실 등에서 필요로 하는 특수효과를 쉽게 프로그래밍 할 수 있는 라이브러리는 개발이 미흡한 실정이다. 게다가 게임을 비롯한 많은 그래픽스 응용 프로그램에서는 OpenGL이나 Direct3D를 이용하여 특수효과를 만들어 내고 있는데도 불구하고 OpenGL 등의 다른 그래픽스 라이브러리에서는 실시간 렌더링을 위한 저수준 연산을 수행하는 API만을 지원하고 있는 실정이기 때문에 프로그램 개발자가 특수효과를 내기 위해서는 저수준 API를 이용하여 코딩하므로 많은 시간과 노력이 소요되며, 필요한 효과의 구현 시 각 기능들을 개별적으로 중복하여 개발하여야 하는 불편함을 감수하여야 한다.

이러한 기존의 방법으로 많은 수의 파티클 즉 입자 오브젝트들을 시뮬레이션하는 것은 현재 3차원 그래픽스 하드웨어의 비약적인 발전에도 불구하고, 다양한 특수효과를 포함하여 실감나는 영상을 보여주는 데 많은 어려움이 있다[7,8].

3. 파티클 시스템

파티클 시스템은 OpenGL 그래픽스 라이브러리에 기반한 특수효과 발생을 위한 상위 수준의 그래픽스 API로 구성되기 때문에 유체속성, 불, 불꽃 속성, 기상현상 등의 여러 가지 특수효과에 대해 설정값의 변경만으로 특수효과 라이브러리를 구축할 수 있게 하며, 게임이나 가상현실 등의 응용 프로그램에서 이를 응용하여 사용할 수 있다.

다음 그림 1은 Game과 가상현실 프로그램 등에서의 응용성과 특수효과 라이브러리 구축을 위한 파티클 시스템 및 OpenGL의 확장성을 보여준다.

본 논문에서는 게임이나 가상현실 응용 프로그램의 실시간 효율성과 프로그래머가 손쉽게 다양한 효과를 만들어낼 수 있는 유연성을 가지고 있으며, 개개의 파라미터에 대해 독립성을 유지하고 확장성이 있는 상위 수준의 파티클 시스템에 대해 API를 구축하여 여러 가지 특수효과의 구현 및 응용을 위한 라이브러리를 구축한다.

파티클 시스템 API의 세부내용은 다음과 같다.

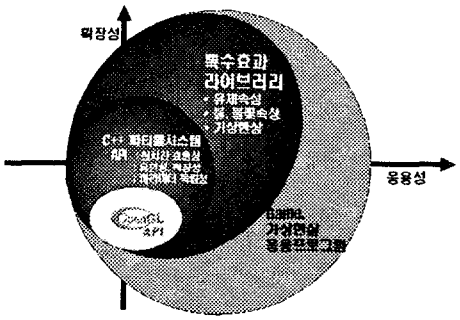


그림 1. 파티클 시스템을 사용한 특수효과 라이브러리의 확장 및 응용

3.1 파티클 속성

파티클 시스템 API 내의 파티클은 위치(position), 속도(velocity), 색상(color), 알파(alpha), 크기(size), 수명(age), 2차 위치(secondary posi-

tion), 2차 속도(secondary velocity) 등의 속성으로 구성한다.

3.2 파티클 함수

API는 파티클 그룹함수, 속성상태 함수, 액션함수, 액션리스트 함수 등 4가지 종류의 함수로 구성한다. 개발할 API 함수 이름은 pFunctionName 형태를 취한다. 프로그래머가 사용하기 쉽고 확장성이 있는 API 함수를 만들기 위하여 다음과 같이 고려한다.

3.2.1 파티클 그룹 함수

모든 파티클들은 같은 힘에 의해 작용하는 파티클들의 집합인 파티클 그룹 내에 존재하며, 몇 개의 파티클 그룹들이 동시에 존재할 수 있으나 모든 API 함수들은 현재의 파티클 그룹에만 적용되며, 모든 액션들은 현재의 파티클 그룹 내의 모든 파티클들에 적용되도록 한다. 파티클 그룹은 개발하게 될 pGenParticleGroups 함수를 이용하여 처음 생성되며, pCurrentGroup 함수를 이용하여 현재의 파티클 그룹으로 전환하게 한다. 파티클 그룹들은 최초에 비어있고 pSource 함수호출에 의해 그룹 내에 새로운 파티클들을 만들어 내도록 한다. 파티클 그룹이 최대 크기에 도달하면 더 이상 파티클의 추가가 불가능하게 한다.

다음 그림 2는 파티클 그룹함수를 나타낸다.

```
void pCopyGroup(...)        //파티클들을 지정한 그룹에서 현재 그룹으로 복사
void pCurrentGroup(...)     //현재 그룹을 바꿈
void pDeleteParticleGroups(...) //하나 이상의 연속된 파티클 그룹을 제거
void pDrawGroup(...)        //각 파티클을 그림
int pGenParticleGroups(...) //파티클 그룹을 생성
int pGetGroupCount()        //현재 그룹내에 존재하는 파티클의 수를 리턴
int pGetParticles(...)      //현재의 그룹에서 메모리로 파티클들을 복사
int pSetMaxParticles(...)   //현재의 그룹내에 파티클들의 최대 수를 바꿈
```

그림 2. 파티클 그룹 내의 파티클 그룹 함수

3.2.2 액션 함수

액션들은 현재의 파티클 그룹 내의 파티클들의 속성을 변경시키는 함수들로 물리적인 힘을 시뮬레이션하거나 각 파티클들이 어떤 특정위치로 이동하게 하는 파라미터 2차 방정식에 의한 경로를 계산하는 함수들로 구성하였다.

액션 함수들은 응용프로그램이 각 장면을 매 프레임마다 렌더링하기 전 또는 렌더링하는 도중에 호출되며, 다음 그림 3과 같이 실행한다.

```
for each particle group j
    pCurrentGroup(j)                    //현재의 파티클그룹 j
    for each time step per render frame //렌더링 프레임마다 반복
        pSource(...)                    //새로운 파티클들을 만들어 낸다.
        other actions....               //다른 액션들을 수행한다.
        pMove()                         //위치를 변경시킨다
    end for
    pDrawGroup(...)                     //파티클들을 그린다.
end for
other drawing...                        //다른 것을 그린다.
```

그림 3. 액션함수의 적용

많은 액션함수들은 파티클이 어떤 영향권에서 멀어짐에 따라 효과를 감쇠시키는 감쇠함수를 필요로 하는데 본 연구에서는 API의 동질성을 위하여 감쇠함수를 필요로 하는 액션함수들에 대하여 다음과 같은 감쇠함수를 사용하였다.

$$f_{m,\varepsilon}(r) = \begin{cases} 0 & r \geq r_{\max} \\ \dfrac{m}{r^2+\varepsilon} & r < r_{\max} \end{cases}$$

여기서 m은 파티클의 크기, r은 거리(반경), rmax는 최대 거리, ε은 거리 r이 0에 가까워짐에 따라 f(r)이 무한대가 되지 않도록 하는 값이다. 다음 그림은 파티클의 크기가 1이고, ε의 값이 1인 이 감쇠함수의 그래프를 나타낸다. 이 함수를 이용함으로써 파티클의 크기와 ε만으로써 더 예민한 감쇠효과를 낼 수 있으며 실행 시 효율성을

크게 증가시키게 된다.

다음 그림 4는 액션함수에서 사용된 감쇠함수에 대한 값의 변화를 그래프로 보여준다.

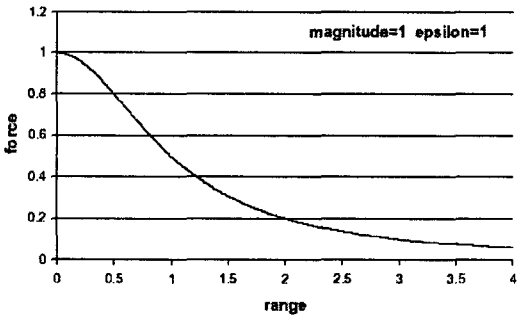


그림 4. 액션함수에서의 감쇠함수 그래프

본 연구에서는 다음 그림 5와 같이 액션함수를 정의하여 API로서 활용하도록 하였다.

3.2.3 액션 리스트 함수

특수효과를 만들어 내는데 필요한 모든 연산들을 캡슐화하기 위하여 필요한 액션들을 모아 놓은 액션 리스트를 사용하고자 한다. 액션 리스트는 특수효과의 추상화를 제공할 뿐 아니라 어떠한 그래픽스 하드웨어에서도 프로그래밍하기 쉬운 인터페이스를 제공해 준다.

액션 리스트 생성함수를 호출하게 되면 어떤 특수효과 생성에 필요한 모든 액션과 상태 변화 호출함수들이 액션 리스트 내에 저장된다. 액션 리스트를 사용함으로써 응용 프로그램과 파티클 동력학을 계산하는 하드웨어 사이의 통신을 줄여주고, 필요한 액션들의 조합을 인식하여 최적화된 함수 호출을 하게 되므로 성능을 크게 향상시킬 수 있다.

```
void pBounce(...)          // 파티클을 공간영역에서 튀어 오르게 함
void pCopyVertexB(...)     //현재 위치로부터 2차 위치를 설정
void pDamping(...)         //파티클 속도를 감쇠
void pExplosion(...)       //폭발
void pFollow(...)          //그룹내의 다음 파티클 쪽으로 가속
void pGravitate(...)       //각 파티클을 다른 파티클 쪽으로 가속화시킴
void pGravity(...)         //일정방향으로 파티클을 가속
void pJet(...)             //분출구 중심 근처에 있는 파티클들을 가속화시킴
void pKillOld(...)         //수명이 다된 파티클을 제거
void pMatchVelocity(...)   //파티클 속도를 그와 인접한 파티클들과 유사하게 변경
void pMove()               //파티클의 위치를 속도에 따라서 이동
void pOrbitLine(...)       //주어진 선상에 가장 가까운 지점으로 가속
void pOrbitPoint(...)      //주어진 중심 지점으로 가속
void pRandomAccel(...)     //파티클들을 임의의 방향으로 가속
void pRandomDisplace(...)  //위치를 영역내의 임의의 다른 위치로 바꿈
void pRandomVelocity(...)  //속도를 영역내에서 다른 속도로 바꿈
void pSink(...)            //정해진 영역을 벗어난 위치의 파티클들을 제거
void pSinkVelocity(...)    //정해진 영역을 벗어나는 속도를 가진 파티클들을 제거
void pSource(...)          //특정영역에 파티클들을 추가
void pSpeedLimit(...)      //각 파티클의 속도를 정해진 최대, 최소 속도로 제한
void pTargetColor(...)     //모든 파티클들의 색상을 특정색상으로 변화
void pTargetSize(...)      //모든 파티클들의 크기를 특정크기로 변화
void pTargetVelocity(...)  //모든 파티클들의 속도를 특정속도로 변화
void pVertex(...)          //하나의 입자를 특정위치에 추가
void pVortex(...)          //파티클들이 선회하도록 함
```

그림 5. 감쇠함수를 필요로 하는 액션 함수들

다음 그림 6은 정의되어 API로 사용되어지는 액션리스트 함수들을 보여준다.

```
void pCallActionList(int action_list_num)
        // 현재의 파티클 그룹에 특정 액션리스트를 적용시킨다.
void pDeleteActionLists(int action_list_num, int ac-tion_list_count=1)
        // 하나 이상의 연속적인 액션리스트를 삭제한다.
void pEndActionList()
        // 새로운 액션리스트 생성을 종료한다.
int pGenActionLists(int action_list_count=1)
        // 빈 액션리스트의 블록을 생성한다.
void pNewActionList(int action_list_num)
        // 특정 액션리스트의 생성을 시작한다.
```

그림 6. 액션리스트 함수

3.3 속성과 영역

새로운 파티클을 생성할 때 파티클의 색상, 속도, 크기, 수명 등의 속성을 지정해 주어야 한다. 대부분의 파티클 효과는 파티클의 초기 속성 값을 변화시키게 된며, 속성들의 변화를 처리하기 위하여 API에 3차원 공간의 범위를 지정해주는 영역을 설정한다.

영역 즉 도메인은 어떤 액션이나 상태함수에 파라미터로 사용되며, 파티클 생성 시에 도메인 내에서 임의로 선택된 위치에서 파티클을 만들어 낸다. 본 연구에서는 구, 평면, 육면체, 실린더, 원뿔, 원반, 삼각형, 직사각형, 선분, 점 등 다양한 형태의 영역을 사용하도록 하였다.

다음 그림 7는 원반 영역 내에서 임의로 선택된 속도 벡터를 나타낸다.

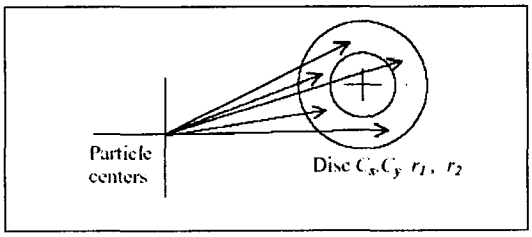


그림 7. 원반영역 내에 임의로 선택된 속도벡터

본 연구에서는 다음 그림 8과 같이 속성 함수들을 정의하였다.

```
void pColor(...)          // 새로운 파티클들의 색상을 지정
void pColorD(...)         // 새로운 파티클들의 색상영역을 지정
void pSize(...)           // 새로운 파티클들의 크기를 지정
void pSizeD(...)          // 새로운 파티클들의 크기영역을 지정
void pStartingAge(...)    // 새로운 파티클들의 수명을 지정
void pTimeStep(...)       // 시간 스텝 길이를 지정
void pVelocity(...)       // 새로운 파티클들의 초속도를 지정
void pVelocityD(...)      // 새로운 파티클들의 초속도 영역을 지정
void pVertexBD(...)       // 새로운 파티클들의 2차 위치영역을 지정
```

그림 8. 속성 함수

4. 파티클 시스템을 이용한 특수효과의 구현

4.1 파티클 시스템과 OpenGL

파티클 시스템은 OpenGL에 기반하여 만들어진 API의 성격을 가지고 있지만 특수효과를 위한 파티클의 생성과 설정, 함수값의 설정은 OpenGL과 독립된 모습을 보여주고 있다. OpenGL은 파티클 시스템의 설정 외에 생성되는 가상세계를 초기화 시켜주며, 장면이 표현되기 위한 윈도우의 생성 및 파티클을 제외한 여러 오브젝트의 생성을 하는 역할을 하고, 파티클 시스템은 입자들의 생성 및 설정을 담당하게 된다.

다음 그림 9는 화면에 디스플레이 되기까지 파티클 시스템과 OpenGL의 상관도를 보여준다.

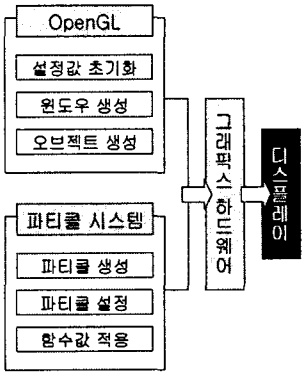


그림 9. 파티클 시스템과 OpenGL

4.2 OpenGL API에서의 파티클 모양 설정

파티클 시스템은 OpenGL에 기반하기 때문에 GL.h 헤더 파일에서 정의된 오브젝트 형태 중의 하나를 파티클의 모양으로 설정하여야 한다. 이때, 적용할 수 있는 오브젝트의 종류는 임의로 선택할 수 있지만, 본 연구에서는 제공되는 여러 가지 오브젝트 중에서 계산이 가장 적게 이루어지는 GL_POINTS를 사용하여 파티클을 표현하도록 하였다.

다음 그림 10은 OpenGL의 GL.h 헤더파일에서 정의된 오브젝트들을 보여준다.

```
#define GL_POINTS                0x0000
#define GL_LINES                 0x0001
#define GL_LINE_LOOP             0x0002
#define GL_LINE_STRIP            0x0003
#define GL_TRIANGLES             0x0004
#define GL_TRIANGLE_STRIP        0x0005
#define GL_TRIANGLE_FAN          0x0006
#define GL_QUADS                 0x0007
#define GL_QUAD_STRIP            0x0008
#define GL_POLYGON               0x0009
```

그림 10. GL.h에서 정의된 오브젝트

4.3 파티클 시스템에서의 도메인 설정

본 연구의 파티클 시스템이 이루어지는 범위를 지정하게 되는 도메인의 설정은 DSphere, PDPlane, PDBox, PDBlob, PDCylinder, PDCone와 같이 다양한 3차원 형태를 가진 도형으로 정의되어 있으며, 나머지 PDTriangle, PDRectangle, PDDisc, PDLine, PDPoint와 같은 형태의 도메인은 2차원으로 정의되어 사용되어진다. 도메인은 어떤 액션이나 상태함수에 파라미터로 사용되기 때문에 파티클의 생성 시에 영역 내에서 임의로 선택된 위치에서 파티클을 생성하는 역할을 한다.

그리고 이중 PDBlob 도메인은 센터의 지정으로 일정 공간에 가우시안 확률밀도에 의한 표준편차를 규정하게 되는데, 이때 도메인 영역 내에 불규칙한 점을 지정하거나 주어진 점이 영역 내에 있는지 판단하게 된다. 이러한 PDBlob 도메인은 생성되는 특수효과가 정규분포를 따르는 것을 정의하기 때문에 시뮬레이션 시 중요한 요소가 된다.

도메인에 대한 정의는 Visual C++ 컴파일러에 헤더 파일로 주어지게 된다.

다음 그림 11은 PDomainEnum 함수의 선언에 따른 파티클 시스템에서의 도메인 정의을 보여준다.

```
PARTICLEDLL_API enum PDomainEnum
{       PDPoint = 0, // Single point
        PDLine = 1, // Line segment
        PDTriangle = 2, // Triangle
        PDPlane = 3, // Arbitrarily-oriented plane
        PDBox = 4, // Axis-aligned box
        PDSphere = 5, // Sphere
        PDCylinder = 6, // Cylinder
        PDCone = 7, // Cone
        PDBlob = 8, // Gaussian blob
        PDDisc = 9, // Arbitrarily-oriented disc
        PDRectangle = 10 // Rhombus-shaped planar region };
```

그림 11. 파티클 시스템에서의 도메인 설정

4.4 파티클의 속성

새로운 파티클의 생성 시에는 색상, 속도, 크기, 수명 등의 속성을 지정해 주어야 하는데, 도메인

```
// State Setting Calls
  ⋮     // 상태설정 함수들의 정의
// Action List Calls
  ⋮     // Action List 함수의 정의
// Particle Group Calls
  ⋮     // Particle Group 함수의 정의
// Actions
  ⋮     // Actions 함수의 정의
```

그림 12. 파티클 속성 함수들의 정의

의 설정이 이루어진 다음에는 상태 설정을 위한 함수들 및 Action List, Particle Group, Actions 함수들을 정의하여 준다.

다음 그림 12는 도메인 설정이 이루어진 후 파티클의 속성을 설정하기 위한 함수들의 정의 과정을 나타낸다.

4.5 특수효과의 구현

다음 그림 13과 14는 본 논문에서 C++로 구현된 파티클 시스템을 토대로 특수효과의 발생을 추상화한 것이다.

이러한 소스 코드는 C++ 언어로 구현된 상위 수준의 API를 호출함으로써 간단히 구현할 수 있으며, 이때 프로그래머는 각 API의 자세한 구현 내용은 모르더라도 액션 함수의 호출과 각 함수에 따른 파라미터 변경만으로 특수효과를 손쉽게 구현할 수 있게 되었다.

```
⋮
void Waterfall(bool first_time = true)
{       pVelocityD(PDBlob, 0.1, 0, 0.1, 0.004);
        pColorD(1.0, PDLine, 0.8, 0.9, 1.0, 1.0, 1.0, 1.0);
        pSize(1.5);
        pStartingAge(0);

        if(first_time)
        {       action_handle = pGenActionLists(1);
                pNewActionList(action_handle);
        }
        pCopyVertexB(false, true);
        pSource(50, PDPoint, -4, 0, 6);
        pGravity(0.0, 0.0, -0.01);
        pKillOld(250);
        pBounce(0, 0.01, 0, PDSphere, -1, 0, 4, 1);
        pBounce(0, 0.01, 0, PDSphere, -2.5, 0, 2, 1);
        pBounce(0, 0.01, 0, PDSphere, 0.7, -0.5, 2, 1);
        pBounce(-0.01, 0.35, 0, PDPlane, 0,0,0, 0,0,1);
        pMove();
        if(first_time)
                pEndActionList(); }
⋮
```

그림 13. 물이 떨어지는 효과 구현의 예

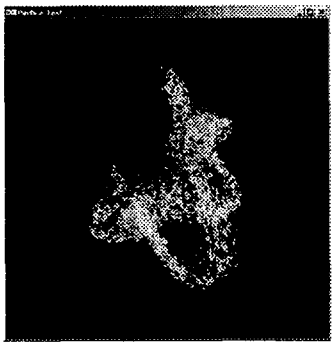
```
⋮
void MovingLights(bool first_time = true)
{       pSize(1.0);
        pVelocityD(PDPoint, 0,0,0);
        pColorD(1.0, PDSphere, .5, .4, .1, .1);
        pStartingAge(0);

        pCopyVertexB(false, true);
        pSource(1, PDBlob, 0, 0, 2, 2);
        pRandomAccel(PDSphere, 0, 0, 0, 0.02);
        pKillOld(20);
        pMove(); }
⋮
```

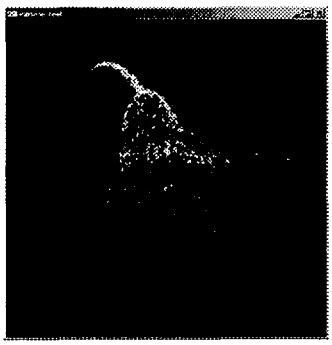
그림 14. 빛 입자가 움직이는 효과 구현의 예

다음 그림 15는 각각 표준 분포를 사용한 파티클 시스템을 이용하여 불꽃과 물이 떨어지는 효과를 구현한 결과를 보여준다.

(a)는 두 개의 중력원 주위를 회전하는 각각 색상이 다른 입자들을 표현하였으며, (b)는 한 장소



(a)



(b)

그림 15. 시뮬레이션된 파티클 시스템

에서 방출되는 파티클들에 대해 두 개의 탄성면에 부딪치게 하여, 파티클의 생명주기 및 이동방향을 설정하여 시뮬레이션하였다.

이들 각각의 그림은 20000개의 파티클을 엘리어스되어진 GL_POINTS를 사용하여 512*512 픽셀의 윈도우 창으로 랜더링하였으며, 본 연구의 구현을 위한 시스템에서 29.7fps에 달하는 초당 프레임 수를 나타내었다.

4.6 구현환경

본 연구의 구현을 위한 소프트웨어와 하드웨어 구현환경은 다음 표 1과 같다.

표 1. 구현환경

소프트웨어	하드웨어
운영체제 : WindowsXP Professional	Graphic Card : Geforce 256DDR
개발환경 : OpenGL, Visual C++	CPU : Athlon 900Mhz

5. 결 론

본 연구에서는 파티클 시스템을 사용하여 객체 단위로 프로그래밍이 가능하고 프로그래머가 구현 알고리즘을 모르더라도 그래픽스의 기본 개념만 이해하면 쉽게 특수효과를 구현할 수 있게 하였다.

기존의 상용 패키지에서 특수효과를 생성하기 위한 파티클 시스템을 지원하고 있지만, 오프라인 상에서만 사용이 가능하여 대화식으로 제작할 수 있는 등의 제한이 있으며, 게임 등의 프로그램에서 특수효과를 내기 위해서는 오프라인 상에서 제작한 특수효과를 사용하거나 자체적으로 특수효과를 내기 위한 기능을 삽입해야 하는데, 그러기 위해서는 개발기간이나 비용이 많이 소요된다는 단점을 가지고 있다.

또한 기존의 다른 연구에서도 유체 속성을 갖는 물질, 불, 불꽃, 폭발 속성을 갖는 물질의 특수효과나 기상현상의 재현을 위한 특수효과 생성 기법을 제시하고 있으나, 모두 특별한 경우의 특수 효과를 내기 위한 개별적인 알고리즘에 관한 것이고, 프로그래머가 게임 등에서 필요로 하는 특수효과를 쉽게 프로그래밍할 수 있는 라이브러리는 개발되어 있지 않다는 단점이 있었다.

본 연구에서 구현된 실시간 동적 파티클 시스템은 게임이나 특수효과를 필요로 하는 애니메이션 응용 프로그램에 직접적인 라이브러리로 활용할 수 있기 때문에 게임 엔진이나 가상현실 등 컴퓨터 애니메이션 응용 프로그램에서 특수효과를 내기 위해 본 연구 결과의 파티클 시스템을 직접 호출하거나 C++로 구현되는 불, 불꽃 속성의 특수효과, 유체속성의 특수효과, 기상현상 재현 특수효과 모듈들을 직접 삽입하여 사용함으로 그 개발 시간과 비용을 크게 줄일 수 있다는 장점이 있다.

이러한 연구결과로 개발된 특수효과 발생 모듈들은 모두 C++ 라이브러리 형태로 제공되므로, 어떠한 플랫폼에서도 쉽게 포팅할 수 있는 유연성을 갖추고 있고 기타 속성함수나 액션함수의 파라미터를 변경함으로써 다양한 특수효과를 발생하는 모듈을 쉽게 개발할 수 있는 등의 많은 이점을 제공하였다.

본 논문에서 구현된 파티클 시스템은 활용성, 유연성 등에서 기존의 라이브러리보다 우수한 성능을 보여주고 있다. 특수효과는 게임이나 영화, 오락 등에서 참여자의 흥미를 크게 증진시키는 효과가 있으므로 이러한 기능을 더욱 확장하여 많은 양의 라이브러리를 구축하여 본격적인 게임 제작을 위한 게임엔진이나 특수효과를 위한 엔진

으로 제공한다면, 엔터테인먼트 산업 등에 상당한 파급효과를 가져올 수 있을 것이다.

감사의 글

본 연구는 2003년도 정보통신부에서 지원하는 기초기술연구지원사업으로 수행되었음.

참 고 문 헌

[1] Reeves, W. T. "ParticleSystems - A Techinigue for Modeling A Class of Fuzzy Objects". Proc. of SIGGRAPH '83, Detroit, Michigan, July, 1983.

[2] 최윤철, 임순범, 고견, "컴퓨터그래픽스 배움터" 생능 출판사, 2003.

[3] McAllister, D. K. "The Design of an API for Particle Systems" http://cs.unc.edu/~davemc/Particle, 1999.

[4] Reeves, W. T. and R. Blau "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems" Proc of SIGGRAPH '85, San Francisco, California, July, 1985.

[5] Leech, J. P. and R. M. Taylor. "Intractive Modeling Using Particle Systems". *Proc of 2nd Conference on Discrete Element Methods*, MIT, 1993.

[6] Allen, M. B. Flow - a particle animation application. http://www.dnai.com/~mba/sortware/flow/, 1999.

[7] http://www.opengl.org

[8] Neider, J. T. Davis, et al. OpenGL Programming Guide. Adison Wesley, 1993.

송 승 헌

- 2000년 순천대학교 이학석사
- 2001년~현재 순천대학교 박사과정, 순천청암대학 겸임교수
- 관심분야 : 컴퓨터그래픽스

김 응 곤

- 1987년 한양대학교 공학석사
- 1992년 조선대학교 공학박사
- 1993년~현재 순천대학교 컴퓨터과학과 교수
- 관심분야 : 컴퓨터그래픽스, 영상처리
- E-mail : kek@sunchon.ac.kr