Journal of Korea Multimedia Society Vol. 11, No. 8, August 2008(pp. 1160-1168)

DMGL: OpenGL ES 기반 모바일 3D 렌더링 라이브러리

황규현†, 박상훈††

요 약

모바일 하드웨어 기술의 비약적인 발전으로 과거에는 실시간으로 렌더링 될 수 없었던 다양한 3D 렌더링 효과들을 모바일 기기 상에서 실시간으로 처리할 수 있게 되었으며, 이를 이용하여 보다 사실적인 모바일 3D 응용 프로그램을 제작할 수 있게 되었다. 본 논문에서는 모바일 환경에서 고화질로 실시간 3D 렌더링을 지원하는 DMGL이라 불리는 플랫폼에 독립적인 OpenGL ES 기반 실시간 모바일 렌더링 라이브러리에 관해 설명한다. 모바일 그래픽스 소프트웨어 개발자들은 이 라이브러리를 이용하여 다양한 고급 실시간 3D 그래픽스 효과들을 간단히 구현할 수 있다. 또한 GPGPU 기반의 라이브러리들은 연기나 불과 같은 자연현상 시뮬레이션을 위한 복잡한 방정식들을 풀고, 그 결과를 실시간 렌더링 할 수 있는 기능을 제공한다.

DMGL: An OpenGL ES Based Mobile 3D Rendering Libraries

Gyu-Hyun Hwang†, Sanghun Park††

ABSTRACT

Recent technological innovations of mobile hardware which make it possible to implement real-time 3D rendering effects under mobile environment have provided a potential to develop realistic mobile application programs. This paper presents platform independent, OpenGL ES based, real-time mobile rendering libraries, called DMGL for supporting high quality 3D rendering on handheld devices. The libraries allows the programmers who develops mobile graphics softwares to generate varying advanced real-time 3D graphics effects without great effort. Moreover, GPGPU-based libraries give a set of functions to solve complex equations for simulating natural phenomena such as smoke and fire, and to render the results in real-time.

Key words: Real-Time Rendering(실시간 렌더링), Mobile 3D Graphics(모바일 3D 그래픽스), OpenGL ES(OpenGL ES), Programmable Graphics Pipeline(프로그래머블 그래픽스 파이프라인), GPGPU(General-Purpose Computing on GPU)

1. 서 론

영상자체의 화질을 중요시 하는 영화, 애니메이션과 달리, 모바일 응용 프로그램에서는 고화질의 영상효과 보다는 실시간 렌더링의 가능 여부가 더 중요한 요소로 고려된다. 과거 GPU(Graphics Processing Unit)가 탑재되어 있지 않던 저성능 모바일 기기들은 범용 컴퓨터 그래픽스 시스템에서 GPU가 담당하는 안티-앨리어싱(anti-aliasing), 텍스쳐 맵핑(texture mapping), 라이팅(lighting)등과 같은 연산을 ARM 계열의 CPU만을 이용하여 소프트웨어 적으로 실행하였다[1]. 따라서 이러한 모바일 기기를 이용하여 범용 PC에서 얻을 수 있는 높은 품질의 영상을 실시간으로 렌더링 하는 것은 불가능 하였고, 텍스트와

※ 교신저자(Corresponding Author) : 박상훈, 주소 : 서울특별시 중구 필동3가 26번지(100-715), 전화 : 02)2260-3765, FAX : 02)2260-3766, E-mail : mshpark@dongguk.edu
접수일 : 2008년 1월 14일, 완료일 : 2008년 7월 21일
† 준회원, 동국대학교 멀티미디어학과 박사과정
(E-mail : spony@dongguk.edu)
†† 종신회원, 동국대학교 영상대학원 멀티미디어학과 조교수
※ 본 논문은 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2006-331-D00497)

2D 스프라이트(sprite)를 이용한 단순한 그래픽스 응용이 주류를 이루었다.

모바일 통신기술의 발달과 함께 HPC(Handheld PCs), PDA(Personal Digital Assistants), 스마트 폰(smart phone) 가운데에서도 GPU가 탑재된 고성능의 모바일 단말기가 출시되면서, 저성능의 CPU에서는 실시간으로 렌더링 될 수 없던 효과들이 GPU를 이용하여 실시간으로 렌더링 가능하게 되었고, 이러한 고성능 모바일 기기의 등장으로 인해 고화질의 모바일 3D 게임과 같은 높은 품질의 콘텐츠에 대한 관심과 수요가 증가되었다. 국내의 경우, 2005년 KTF GPANG, SKT GXG 서비스 시작과 고정 그래픽스 파이프라인(fixed graphics pipeline)을 지원하는 GPU가 탑재된 모바일 단말기의 보급으로 고화질 렌더링이 가능한 모바일 계산 환경이 구축되었다. 그럼에도 불구하고, 고정 그래픽스 파이프라인을 지원하는 모바일 단말기들이 갖는 그래픽스 파이프라인 구조상의 한계로 인해 많은 계산이 요구되는 특수한 렌더링 효과들을 높은 품질로 실시간 렌더링 하는 것은 불가능하였다. 해외의 경우, 최근 OpenGL ES 2.0 프로그래머블 그래픽스 파이프라인(programmable graphics pipeline)을 지원하는 GoForce5500 GPU가 탑재된 O2 XDA Flame[2]과 같은 고성능 모바일 단말기의 출시로, 모바일 기기에서도 버텍스 쉐이더(vertex shaders)와 프래그먼트 쉐이더(fragment shaders)를 이용하여 고화질 영상의 실시간 렌더링이 가능하게 되었다. 그러나 아직까지 이동통신사에 의해 제공되는 대부분의 상용 모바일 3D 솔루션들은 프로그래머블 그래픽스 파이프라인을 지원하는 OpenGL ES 2.0을 지원하지 않고 OpenGL ES 1.0만을 지원하기 때문에 많은 계산이 요구되는 그래픽스 효과들을 높은 품질로 실시간 렌더링 하는 것이 쉽지 않은 상황이다.

본 논문에서는 3D 모델을 모바일 환경에서 높은 품질로 실시간 렌더링하고 다양한 특수 효과들을 모바일 3D 응용에 실시간으로 적용할 수 있도록 설계된, 플랫폼에 독립적인 OpenGL ES 기반의 실시간 렌더링 라이브러리에 대해 설명한다. DMGL(Dongguk Mobile Graphics Libraries)이라는 이름으로 개발된 라이브러리는 고정 그래픽스 파이프라인과 프로그래머블 그래픽스 파이프라인 구조에서 각각 구현되었기 때문에, 어떤 그래픽스 파이프라인을 지원하는가에 관계없이 모든 그래픽스 가속 기능을 제공하는 모바일 단말기에서 사용가능하도록 설계되었다. 고정 그래픽스 파이프라인을 지원하는 OpenGL ES 1.1 기반의 라이브러리는 투영된 조명(projective lighting), 투영된 그림자(projective shadow), 빌보드(billboard), 환경 맵핑(environment mapping), 범프 맵핑(bump mapping), 안개(fog)와 같은 효과에 대한 실시간 렌더링을 지원한다. 또한 프로그래머블 그래픽스 파이프라인을 지원하는 OpenGL ES 2.0 기반의 라이브러리는 프로그래머블 쉐이더를 이용하여 GPU상에서 유체의 움직임을 실시간 시뮬레이션하는 기능을 제공한다.

2. 연구 내용

그래픽스 파이프라인은 입력된 기하 프리미티브(geometric primitive)에 대해 버텍스 단위 연산(per-vertex operation), 프리미티브 조립(primitive assembly), 래스터화(rasterization), 프래그먼트 단위 연산(per-fragment operation)을 순차적으로 수행하여 계산된 최종 색을 화면에 출력하는 하드웨어 또는 시스템 구조를 의미한다[3]. OpenGL ES 그래픽스 파이프라인은 기본 구조에 따라 고정 그래픽스 파이프라인과 프로그래머블 그래픽스 파이프라인으로 나누어진다[4]. 고정 그래픽스 파이프라인에서 버텍스 단위 연산은 입력으로 들어오는 버텍스의 좌표와 법선(normal) 벡터를 이용하여 조명(illumination)을 계산하고, 계산된 조명과 재질(material), 모델-뷰 행렬(model-view matrix)을 곱한 버텍스의 좌표를 프리미티브 조립 단계로 넘겨주는 역할을 한다. 버텍스 단위 연산 단계에서 프리미티브 조립 단계로 전달된 정점의 좌표, 조명, 재질들은 래스터화 단계를 거쳐 프래그먼트 단위 연산 단계로 전달되고 이를 통해 최종적으로 화면에 그려질 영상이 프레임 버퍼에 저장된다[5]. 프로그래머블 그래픽스 파이프라인은 고정 그래픽스 파이프라인과 달리 정점 단위 연산과 프래그먼트 단위 연산 부분을 프로그래머가 제어할 수 있으며, 이를 이용함으로써 기존 고정 그래픽스 파이프라인에서 구현하기 어려운 특수한 효과들을 응용 프로그램에서 실시간으로 디스플레이 할 수 있다[6].

본 논문에서는 모바일 환경에서 실시간 렌더링이

요구되는 응용 프로그램의 제작에 활용될 수 있는 라이브러리를 제안한다. 고정 그래픽스 파이프라인을 지원하는 OpenGL ES 1.0만을 기반으로 제작된 상용 모바일 3D 솔루션과 달리, DMGL은 고정 그래픽스 파이프라인을 지원하는 OpenGL ES 1.1과 프로그래머블 그래픽스 파이프라인을 지원하는 OpenGL ES 2.0을 기반으로 각각 제작되었으며, 그림 1과 같은 계층 구조의 일부 모듈(module)로 이용될 수 있다. 즉, 미디어 엔진(media engine) 계층과 하드웨어 엔진(hardware engine) 계층 위에 존재하며 게임 엔진(game engines)과 미들웨어 라이브러리(middleware libraries)와 함께 응용 프로그램 제작에 활용될 수 있다. OpenGL ES 1.1을 기반으로 제작된 실시간 렌더링 라이브러리를 이용하여 모바일 게임에 투영된 조명, 투영된 그림자, 빌보드, 환경 맵핑 효과 등을 적용할 수 있다. 또한, 기존의 상용 모바일 3D 솔루션에서 높은 품질로 렌더링 불가능 했던, 불규칙한 연기와 같은 많은 계산이 요구되는 실시간 시뮬레이션 효과들은 OpenGL ES 2.0을 기반으로 제작된 실시간 GPGPU 라이브러리를 이용하여 계산되고 렌더링 될 수 있다.

2.1 고정 그래픽스 파이프라인 기반 라이브러리

그림 2는 OpenGL ES 1.1 고정 그래픽스 파이프라인을 기반으로 제작된 실시간 렌더링 라이브러리에 포함된 함수들을 사용하여 구현된 3D 렌더링 예제 프로그램들을 보여준다. 프로그래머들은 간단한 함수 호출을 통해 이와 같은 효과들을 모바일 기기에서 실시간 렌더링으로 표현할 수 있다.

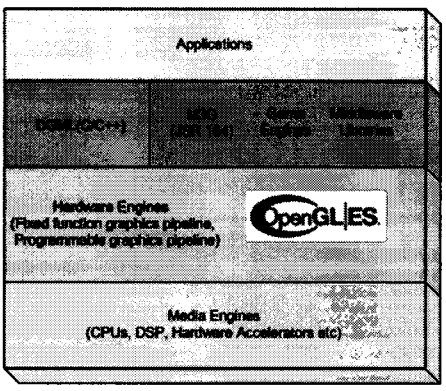


그림 1. DMGL과 모바일 단말환경의 계층구조

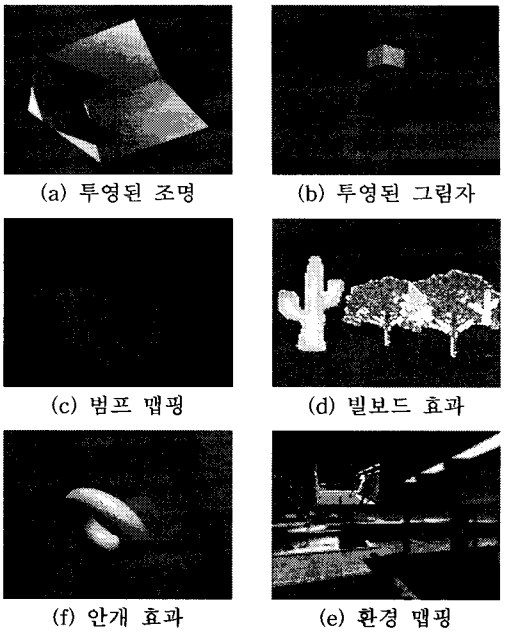


그림 2. OpenGL ES 1.1 기반 라이브러리를 이용한 실시간 응용의 예

투영된 조명은 슬라이드 프로젝터를 이용해 영상을 스크린에 투영하듯 2차원 텍스쳐를 조명의 위치에서 3차원 물체 표면에 투영하여 스포트라이트(spotlights)와 같은 조명 효과를 표현하는 기법[7]이다. 광원의 방향과 컷오프 각(cutoff angle)을 이용하여 스포트라이트 효과를 표현하는 퐁의 조명 모델(Phong lighting model)과 달리, 조명 중심의 변환을 거쳐 얻은 텍스쳐 좌표만을 사용하여 스포트라이트 효과를 표현한다. 이때 사용되는 텍스쳐 좌표는 식(1)과 같이 3D 모델의 객체 공간(object space)에서의 좌표에 조명의 시점 행렬(view matrix)과 조명의 투영 행렬(projection matrix)을 곱하여 얻은 결과를 사용하였다. 이 효과는 객체의 움직임과 조명의 움직임을 계산하는 dgTrackMotion()과 dgMakeBorder(), dgLoadProjectionTexture()등의 함수 호출을 통해 간단히 수행된다.

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} 0.5 & & & 0.5 \\ & 0.5 & & 0.5 \\ & & 0.5 & 0.5 \\ & & & 1 \end{bmatrix} \begin{bmatrix} Light\ Frustum \\ (projection) \\ Matrix \end{bmatrix} \begin{bmatrix} Light\ View \\ (lookat) \\ Matrix \end{bmatrix} \begin{bmatrix} Modeling \\ Matrix \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix} \quad (1)$$

그림자 효과는 현실감 있는 영상을 생성하고 사용자로 하여금 장면에 배치된 물체간의 위치 관계를

알 수 있게 해주는 렌더링 효과이다[8]. 투영된 그림자 효과의 기본 알고리즘은 카메라를 조명의 위치로 이동하고 조명과 같은 방향을 바라보게 설정한 다음, 3D 물체를 평면에 투영하여 얻은 영상을 그림자로 사용하는 기법이다. 식 (2)와 같이 객체공간의 3D 물체를 평면에 투영하는 조명 중심의 변환 행렬이 적용된 영상을 그림자로 사용하였다. 여기서 *Light Frustum Matrix*는 조명의 투영 행렬이고 *Light View Matrix*는 조명의 시점 행렬이다. 이 기능은 dgLoadProjectionTexture(), dgDrawObject(), dgMakeBorder(), 함수로 구현되었다.

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = \begin{bmatrix} Light\ Frustum \\ (projection)\ Matrix \end{bmatrix} \begin{bmatrix} Light\ View \\ (lookat)\ Matrix \end{bmatrix} \begin{bmatrix} Modeling \\ Matrix \end{bmatrix} \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix} \quad (2)$$

낮은 다각형의 균일한 표면을 굴곡이 있는 것처럼 보이도록 하는 렌더링 효과를 범프 맵핑[9]이라 하며 색깔 대신 법선 벡터를 저장하고 있는 노말 맵(normal map)을 텍스쳐로 사용하여 맵핑된 위치의 법선 벡터를 대치하고 조명모델을 적용한다. 본 논문에서 범프 맵핑 효과는 dgGenBump(), dgCombineImage() 라는 이름의 함수로 구현하였다. 빌보드 기법(billboarding)은 시야 방향을 기반으로 하여 다각형의 방향을 결정하는 것을 말하며[5], 빌보드의 기법으로는 화면 정렬(screen-aligned) 빌보드와 월드 중심(world-oriented) 빌보드, 축(axial) 빌보드 기법이 있으며, 본 논문에서 화면 정렬 빌보드 방법을 dgGenBillboard(), dgTexGen() 이라는 이름의 함수로 구현하였다.

반사 맵핑(reflection mapping)이라고도 불리는 환경 맵핑은 곡면에서의 반사 계산을 근사화 하여 물체에 주위 환경이 반사되어 보이는 것과 같은 효과를 얻기 위한 기법이다. Blinn과 Newell이 제안한 환경 맵핑 방법[10]은 식 (3)과 같이 노말 벡터 n과 카메라에서 물체로 향하는 벡터 e를 이용하여 반사되는 벡터 r을 구한 뒤, 식 (4)에서와 같이 반사 벡터 r로부터 ρ와 ϕ를 계산하고, 이를 텍스쳐 좌표로 사용한다.

$$r = e - 2(n \cdot e)n \quad (3)$$

$$\rho = cos^{-1}(r_z), \quad \phi = tan^{-1}\left(\frac{r_y}{r_x}\right) \quad (4)$$

하지만 이러한 방법은 $\phi = 0$ 인 곳에서 경계선이 존재하고 맵이 극점에서 수렴하며 상단이나 하단 모서리 부분에서 왜곡이 발생하는 문제점을 가지기 때문에, 본 논문에서는 균일한 샘플링 특성을 갖고 실시간 렌더링이 가능한 Greene의 방법[11]을 이용하였으며, 이 계산은 dgGeMatrix(), dgBackground_gen(), dgComputationCubemap() 함수로 구현하였다.

2.2 프로그래머블 그래픽스 파이프라인 기반 라이브러리

프로그래머블 그래픽스 파이프라인을 기반으로 제작된 라이브러리는, 고정 그래픽스 파이프라인만을 기반으로 제작된 상용 모바일 3D 솔루션과 달리, 프래그먼트 쉐이더를 이용하여 유체 시뮬레이션의 대표적인 수식인 Navier-Stokes 방정식을 GPGPU를 이용해 풀이할 수 있다. 뿐만 아니라, 라이브러리를 이용하여 그림 3과 같이 3D 모델의 움직임에 영향

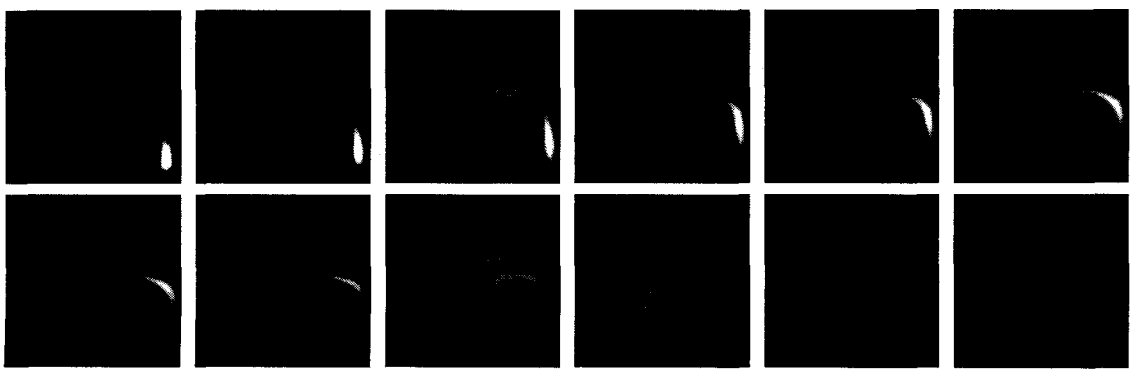


그림 3. 3D 물체의 움직임에 따른 연기의 흐름 변화(원숭이 머리 모델이 오른쪽에서 왼쪽으로 회전하는 상황을 프레임 별로 캡쳐)

을 받는 유체의 움직임을 실시간으로 표현할 수 있다.

2.2.1 Navier-Stokes 방정식을 이용한 연기 시뮬레이션

연기, 불과 같이 불규칙한 유체의 움직임을 자연스럽게 표현하기 위한 연구가 컴퓨터 그래픽스 분야에서 활발하게 수행되고 있다. 유한 차분(finite differential)형태의 Navier-Stokes 방정식 풀이를 통한 유체 시뮬레이션 방법이 Foster와 Metaxas에 의해 소개되었지만 시간 간격(time-step)이 큰 경우 안정적인 해를 구할 수 없는 문제점을 가지고 있었다[12]. 이러한 문제점을 해결하기 위해 Stam은 시간간격이 큰 경우에도 안정적인(stable)해를 얻을 수 있는 세미-랑그랑주 방법(semi-lagrangian method)에 의한 효과적인 풀이 방법을 소개 하였다[13]. Navier-Stokes 방정식에 각 유체마다의 특성을 고려하여 적절한 변형을 적용함으로써 연기, 불, 물 등의 자연현상을 시뮬레이션 할 수 있다. 아래 식 (5)는 앞에서 언급한 불규칙한 유체를 시뮬레이션 하기 위한 Navier-Stokes 방정식이다.

$$\frac{\partial u}{\partial t}=-(u\cdot\nabla)u-\frac{\nabla p}{\rho}+\nu\nabla^2u+f \qquad (5)$$
$$\nabla\cdot u=0$$

식 (5)의 $u(x,t)$는 위치벡터 x의 시간 t에서의 유체의 속도장(velocity field)이고 이를 시간에 대해 미분한 것이 $\frac{\partial u}{\partial t}$이다. 여기에서 p는 유체의 압력(pressure)이고 ρ는 밀도(density), ν는 점성계수(viscosity) 이며 f는 외부에서 주어지는 힘(external force)이다[13]. 연기와 같은 기체를 표현하는 경우, Navier-Stokes 방정식의 기본 형태인 식 (5)는 점성이 0인 다음과 같은 식 (6)으로 변형이 가능하다.

$$\frac{\partial u}{\partial t}=-(u\cdot\nabla)u-\frac{\nabla p}{\rho}+f,\quad \nabla\cdot u=0 \qquad (6)$$

이와 같은 비점성 유체를 표현하는 수식을 오일러 방정식(Euler equation)이라 부르며, 입자(particle)에 기반을 둔 라그랑지안 방식[14]과는 다르게 고정된 격자 공간을 기반으로 유체의 움직임을 표현하게 된다. 라그랑지안 기법을 이용한 유체 시뮬레이션은 오일러 기법 보다 메모리를 효율적으로 관리할 수 있지만 이웃한 입자를 찾는데 많은 시간을 소비하기 때문에, 본 논문에서는 모바일 환경에서 실시간으로 렌더링 하기 위해 격자 기반의 오일러 방정식을 사용하였다.

2.2.2 Navier-Stokes 방정식 풀이

오일러 방정식을 사용하여 비점성 유체를 시뮬레이션 하기 위해 Stam이 소개한 격자 모델을 사용하였고, 연산 속도의 향상을 위해 프로그래머블 쉐이더를 이용하여 복잡한 시뮬레이션 계산을 CPU가 아닌 GPU상에서 수행하였다. 유체의 속도 벡터는 밀도, 온도 등의 물리량을 이동시키는데 이것을 확산(advection)이라 부르며, 현재 격자에 위치한 입자의 시간 간격 Δt 이전의 격자 위치에서 속도 정보를 가지고 현재 격자의 값을 다시 계산하는 과정을 의미한다[15]. 아래 식 (7)은 속도 u에 대한 확산을 표현하는 수식이다.

$$u(\vec{x},t+\Delta t)=u(\vec{x}-u(\vec{x},t)\Delta t,t) \qquad (7)$$

위의 식 (7)을 시간 t에 관하여 양변을 편미분 하면 식 (8)과 같은 확산 방정식을 얻을 수 있다.

$$u_t=-u\bullet\nabla u \qquad (8)$$

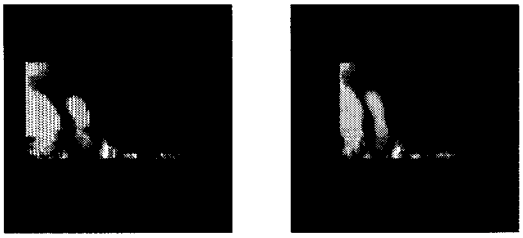
외부의 힘과 확산 단계를 통해 계산된 값이 발산되는 것을 막기 위해 오일러 방정식의 질량 보존식의 Helmholtz-Hodge 분해를 식 (9)와 같은 속도에 관한 식으로 표현할 수 있다[15].

$$\nabla^2p=\nabla\bullet u \qquad (9)$$

$\nabla^2x=b$인 경우를 포아송 방정식(Poisson equation)이라 하며 확산단계에서 얻은 속도를 이용하여 위의 방정식으로부터 압력을 구한 후, 각 격자의 속도와 압력을 갱신하고 이웃한 격자 사이의 발산을 막기 위한 보정(correction)과정을 되풀이 한다.

2.2.3 유체 시뮬레이션 결과의 렌더링

유체 시뮬레이션 결과의 렌더링을 위해 쉬어-왑 볼륨렌더링(shear-warp volume rendering)을 이용한 3차원 연기 표현 기법을 사용할 수 있다. 이 기법은 사실적인 연기의 흐름을 표현할 수 있는 장점이 있지만, 낮은 초당 프레임 수와 많은 메모리를 요구하는 단점이 있다. 그림 4는 OepnGL ES 1.1 기반 실시간 렌더링 라이브러리를 이용하여 구현된 3차원 연기 시뮬레이션 결과를 저장한 볼륨데이터에 쉬어-



(a) 쉬어 단계 후 (b) 왑 단계 후

그림 4. 쉬어-왑 볼륨렌더링을 적용한 3차원 연기

왑 알고리즘을 적용한 결과 영상이다. 실험에 사용된 시뮬레이션 격자 해상도는 32×32×32이며, 초당 1.5 프레임의 렌더링 속도를 얻을 수 있었다.

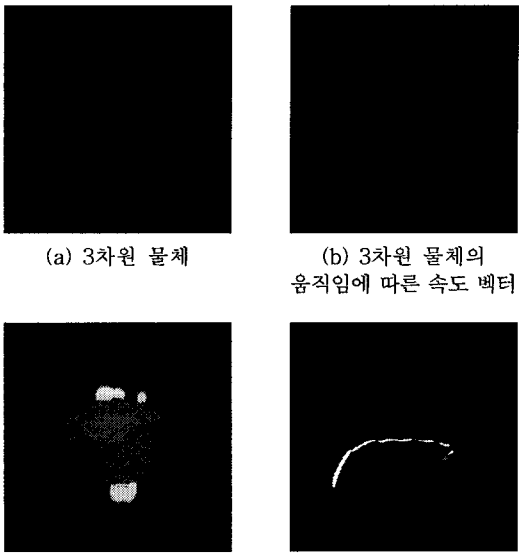
쉬어-왑 볼륨 렌더링 기법이 갖는 낮은 렌더링 속도와 많은 메모리 요구의 문제를 해결하기 위해, 본 논문에서는 실시간 렌더링이 중요하게 고려되는 모바일 환경에 적합한 새로운 유체 시뮬레이션 렌더링 기법을 개발하였다. 이것은 버텍스 쉐이더를 이용하여 2차원 연기를 표현하는 방법으로, 쉬어-왑 볼륨렌더링을 이용한 3차원 연기 표현방법에 비해 낮은 품질의 영상을 얻게 되는 단점이 있으나, 계산에서 요구되는 메모리의 크기가 작을 뿐만 아니라, 물체의 움직임에 따른 연기의 흐름을 실시간으로 렌더링 할 수 있다는 매우 중요한 장점이 갖는다.

버텍스 쉐이더를 이용한 2차원 연기 표현은 3차원 물체의 움직임에 따른 각 버텍스의 속도 벡터를 버텍스 쉐이더를 이용하여 계산하고, 3차원 물체의 움직임에 따라 유도되는 연기의 흐름을 표현하는 방법이다. 식 (10)은 3차원 물체의 움직임에 따른 각 버텍스의 속도 벡터를 구하기 위해 본 논문에서 적용한 수식으로 p는 Δt 이전 클립 좌표계(clip coordinates)에

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix} = \begin{bmatrix} Previous \\ Model-View \\ Projection \\ Matrix \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix},$$

$$\begin{bmatrix} c_x \\ c_y \\ c_z \\ c_w \end{bmatrix} = \begin{bmatrix} Current \\ Model-View \\ Projection \\ Matrix \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}, \quad (10)$$

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ v_w \end{bmatrix} = \left(\begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \\ c_w \end{bmatrix} \right) \times \frac{1}{\Delta t}$$



(a) 3차원 물체 (b) 3차원 물체의 움직임에 따른 속도 벡터

(c) 3차원 물체의 움직임에 따른 속도장 (d) 최종 영상

그림 5. 3차원 물체의 움직임에 따라 유도되는 2차원 연기

서의 버텍스 좌표이고 c는 현재 클립 좌표계에서의 버텍스 좌표이다. v는 p와 c의 차이를 Δt로 나눈 결과로 3차원 물체의 버텍스 별 속도 벡터가 된다.

그림 5는 위의 수식을 통해 계산된 3차원 물체의 버텍스 별 속도 벡터를 이용하여 확산, 사영 단계를 거쳐 생성된 속도장을 보여주고 있다. 제안된 버텍스 쉐이더를 이용한 수식을 적용한 실험에서, 64×64 시뮬레이션 격자 해상도에서 초당 26 프레임의 렌더링 속도를 얻을 수 있었다.

3.실험 결과 및 성능 분석

개발된 실시간 렌더링 라이브러리는 Microsoft Visual Studio 2005, Embedded Visual C++ 4.0, PocketPC 2003 에뮬레이터와 nVIDIA OpenGL ES 2.0, Vincent 3D Rendering, OpenKODE 라이브러리를 이용하여 개발 되었으며, 실험에 사용된 PC는 Intel Core2 6600, 2GB 메모리, GeForce 8800GT (256MB) GPU를 탑재하고 있다.

3.1 OpenGL ES 1.1 기반 실시간 렌더링 라이브러리

OpenGL ES 1.1 기반 실시간 렌더링 라이브러리를 이용하여 구현된 빌보드, 환경 맵핑, 범프 맵핑,

표 1. 실시간 렌더링 라이브러리 효과별 초당 프레임 수(P.L: 투영된 조명, P.S: 투영된 그림자)

실시간 렌더링 효과	빌보드	환경 맵핑	범프 맵핑	P.L	P.S	안개
초당 프레임 수	25	18	50	25	50	50

투영된 조명, 투영된 그림자, 안개 효과들을 위한 함수들은 모바일 3D 응용 프로그램의 일부로서 실시간에 수행된다. 표 1은 구현된 렌더링 라이브러리를 이용한 실험에서 얻어진 렌더링 효과별 초당 프레임수를 나타낸 것으로 환경 맵핑을 제외한 렌더링 효과들은 모바일 환경에서 초당 25 프레임 이상의 속도로 실시간 렌더링이 가능함을 확인할 수 있다. 환경 맵핑의 경우 초당 18 프레임의 속도를 얻을 수 있었는데, 그 이유는 모델링 변환에 따른 3D 물체의 각 버텍스 별 법선벡터와 반사 벡터, 환경 맵핑에 사용되는 텍스쳐 좌표 등의 계산 과정을 소프트웨어적으로 처리하기 때문이다. 이러한 속도의 한계를 극복하기 위해, 환경 맵핑은 OpenGL ES 2.0 기반 라이브러리에서 다시 구현이 되었으며, 쉐이더 프로그래밍을 통해 초당 30 프레임 이상의 속도를 얻을 수 있었다.

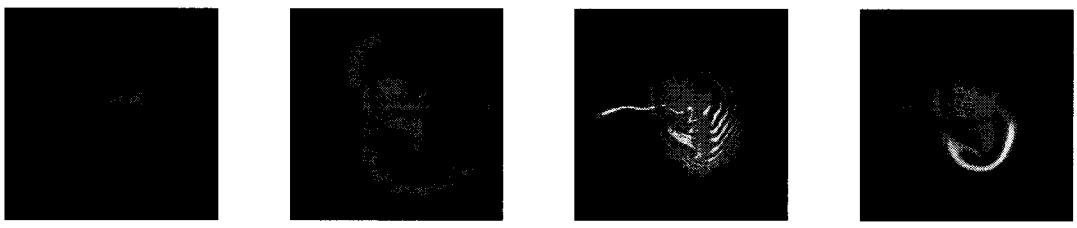
그림 6. 실시간 렌더링 라이브러리의 여러 가지 함수들을 동시에 적용한 예

그림 6은 실시간 렌더링 라이브러리의 효과들을 응용한 예로서, 1,066개의 다각형으로 구성된 3D 모델과 상자 배경에 대해 투영된 그림자와 범프 맵핑을 각각 적용한 결과이다. 3D 모델의 움직임과 조명의 움직임에 따라 투영된 그림자가 실시간으로 반영되며, 초당 25 프레임 이상의 속도를 얻을 수 있었다.

3.2 OpenGL ES 2.0 기반 실시간 렌더링 라이브러리

OpenGL ES 2.0 기반 렌더링 라이브러리를 이용하여 많은 계산이 요구되는 렌더링 효과들을 GPU상에서 수행할 수 있으며, 이 라이브러리를 이용하여 연기와 같은 유체의 움직임을 실시간으로 표현할 수 있다. 그림 7은 시뮬레이션 격자 해상도의 변화에 따른 결과 영상의 차이를 비교한 것이다. 3D 모델의 움직임으로부터 속도장을 생성하고 3D 모델의 움직임에 따라 변화되는 2D 연기를 표현한 것이다. 실험에 사용된 3D 모델은 2,904개의 다각형으로 이루어져 있으며, 이 실험에서는 포아송(Poisson) 반복 횟수를 20회로 고정하였다. 시뮬레이션 된 연기는 32×32, 64×64 시뮬레이션 격자 해상도에서 각각 초당 47.35, 26.26 프레임의 속도로 렌더링 되며, 모바일 시스템의 성능과 화면 크기를 고려할 때, 그 이상의 시뮬레이션 해상도를 이용할 필요는 없을 것으로 판단된다. 렌더링 성능을 비교하기 위해 [16]에서 소개된 NF3D 기반으로 구현된 유체 시뮬레이션 실험의 격자 해상도별 초당 프레임 수를 참고하였다. NF3D 기반의 실험 결과에 따르면 32×32, 64×64 시뮬레이션 격자 해상도에의 렌더링 성능은 각각 초당 약 27, 18 프레임이다. 따라서 본 논문에서 제안한 버텍스 쉐이더을 이용한 연기 시뮬레이션 방법이 NF3D 기반으로 구현된 유체 시뮬레이션의 결과보다 좋은 화질로 더 빠르게 렌더링 가능함을 확인 할 수 있었다.

표 2는 64×64 시뮬레이션 격자 해상도에서 포아



(a) 32×32 (b) 64×64 (c) 128×128 (d) 256×256

그림 7. 시뮬레이션 격자 해상도 변화에 따른 결과 영상(포아송 반복 횟수: 20)

(a) 20회 (a) 40회 (a) 60회 (a) 80회

그림 8. 포아송 반복 횟수에 따른 결과 영상(시뮬레이션 격자 해상도: 64×64)

표 2. 포아송 반복 횟수에 따른 시뮬레이션 속도 (격자 해상도 : 64×64)

포아송 반복 횟수(회)	20	40	60	80
DMGL을 사용한 경우 초당 프레임수(fps)	26	18	14	11
DMGL을 사용하지 않은 경우 초당 프레임수(fps)	15	11	8	6

송 반복 횟수에 따른 시뮬레이션 속도 변화를 나타낸 것으로, 동일한 조건에서 본 논문에서 제안하는 렌더링 라이브러리를 사용한 경우와 그렇지 않은 경우의 초당 프레임 수를 비교하고 있다. 그림 8에서 보는 것과 같이 포아송 반복 횟수가 높을수록 정밀한 연기의 움직임을 시뮬레이션 할 수 있다. DMGL을 사용한 경우, 64×64의 시뮬레이션 격자 해상도에서 20~25회 정도의 포아송 반복 횟수를 적용했을 때 실시간으로 렌더링이 가능함을 확인 할 수 있다.

4. 결 론

모바일 하드웨어 기술의 비약적인 발전으로 과거에는 실시간 수행을 기대할 수 없었던 다양한 그래픽스 효과들이 모바일 환경에서 실시간 구현 가능하게 되었다. 또한 고정 그래픽스 파이프라인을 지원하는 모바일 단말기에서 실시간 연산이 불가능했던, 높은 계산 비용이 요구되는 특수 효과들도 프로그래머블 쉐이더를 지원하는 모바일용 GPU의 개발로 실시간 렌더링이 가능하게 되었다.

본 논문에서는 모바일 환경에서 3D 모델을 높은 품질로 실시간 렌더링 하고, 유체의 흐름과 같은 다양한 특수 효과들을 실시간에 표현할 수 있는 OpenGL ES 기반의 렌더링 라이브러리에 대해 설명하였다. 이 라이브러리는 고정 그래픽스 파이프라인과 프로그래머블 그래픽스 파이프라인 구조에서 각각 구현되었으며, 두 가지 그래픽스 파이프라인을 지원하는 모든 모바일 단말기를 지원하도록 설계되었다. 고정 그래픽스 파이프라인을 지원하는 OpenGL ES 1.1을 기반으로 개발된 라이브러리를 이용하여 투영된 조명, 투영된 그림자, 빌보드, 환경 맵핑, 범프 맵핑, 안개와 같은 실시간 렌더링 효과를 적용할 수 있다. 또한, OpenGL ES 2.0 기반의 라이브러리는 많은 계산이 요구되는 연기나 불과 같은 자연현상을 GPU상에서 시뮬레이션하고 렌더링 할 수 있는 기능을 제공한다. 이를 이용하여 구현된 유체 시뮬레이션 효과는 하드웨어 성능의 제약으로 인해 아직까지 3차원으로 확장하기에는 무리가 있지만, 모바일 하드웨어의 발전 속도로 미루어 볼 때 향후 몇 년 이내에 3차원 유체의 움직임도 무리 없이 실시간으로 표현할 수 있을 것으로 기대한다.

참 고 문 헌

[1] D. Astle, and D. Durnil, *OpenGL ES Game Development*, Muska & Lipman Publishing, Boston, MA., 2004.

[2] nVIDIA, http://nvdeveloper.nvidia.com, 2007.

[3] A. Malizia, *Mobile 3D Graphics*, Springer-Verlag New York Inc, New York, NY., 2007.

[4] The Khronos Group, http://www.khronos.org, 2007.

[5] T. Akenine-Moller, *Real-Time Rendering*, AK PETERS, Wellesley, MA., 2003.

[6] A. Boreskov, *Developing and Debugging Cross-Platform Shaders*, A-List Publishing, Wayne, PA., 2007.

[7] T. McReynolds, D. Blythe, B. Grantham, and

S. Nelson, "Advanced Graphics Programming Techniques Using OpenGL," *SIGGRAPH '99 Course Note*, 1999.

[8] R. Fernando, and M. Kilgard, *The CG Tutorial : The Definitive Guide to Programmable Real-Time Graphics*, Addison-Wesley, Indianapolis, Indiana, 2003.

[9] J. Blinn, "Simulation of wrinkled surface," *In Proceedings of SIGGRAPH '78*, pp. 286-292, 1978.

[10] J. Blinn and M. Newell, "Texture and reflection in computer generated images," *Communication of the ACM*, Vol.19, No.10, pp. 542-547, 2001.

[11] N. Greene, "Environment Mapping and Other Applications of World Projections," *IEEE Computer Graphics and Applications*, Vol.6, No.11, pp. 21-29, 1986.

[12] N. Foster and D. Metaxas, "Modeling the Motion of a Hot, Turbulent Gas," *In Proceedings of SIGGRAPH '97*, pp. 181-188, 1997.

[13] J. Stam, "Stable Fluids," *In Proceedings of SIGGRAPH '99*, pp. 121-128, 1999.

[14] M. Muller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," *In Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 154-159, 2003.

[15] M. Harris, "Fast Fluid Dynamic Simulation on the GPU," *GPU Gems*, pp. 637-664, Addison-Wesley, Indianapolis, Indiana, 2004.

[16] 우상혁, 조미리나, 박동규, "모바일 환경에서 불꽃의 실시간 시뮬레이션과 렌더링," 멀티미디어학회논문지, 제10권, 제7호, pp. 934-943, 2007.

황 규 현

2006년 2월 대구대학교 멀티미디어공학과 졸업(공학사)
2008년 2월 동국대학교 멀티미디어학과 졸업(공학석사)
2008년 3월~현재 동국대학교 멀티미디어학과 박사과정

관심분야 : 컴퓨터 그래픽스, 모바일 3D 그래픽스, 가상현실 등.

박 상 훈

1993년 8월 서강대학교 수학과 졸업(이학사)
1995년 8월 서강대학교 컴퓨터학과 졸업(공학석사)
2000년 2월 서강대학교 컴퓨터학과 졸업(공학박사)
2000년 3월~2000년 6월 서강대학교 컴퓨터학과 박사후 연구원
2000년 7월~2002년 8월 University of Texas at Austin 박사후연구원
2002년 9월~2005년 2월 대구가톨릭대학교 컴퓨터정보통신공학부 조교수
2005년 3월~현재 동국대학교 영상대학원 멀티미디어학과 조교수
관심분야 : 컴퓨터 그래픽스, 과학적 가시화, 가상현실, 모바일 컴퓨팅, 컴퓨터 게임 등.