'92 韓國自動制御學術會議論文集 1992. 10. 19~21

Simulation을 위한 고성능 그래픽 시스템의 개발에 관한 연구

노갑선, 박재현, 장래혁, 박정우, 구경훈, 이재영, 권욱현

서울대학교 공과대학 제어계측공학과

A Study on the Development of High Performance Graphics System for Simulation

Gab Seon Rho, Jaehyun Park, Naehyuck Chang, Jung Woo Park, Kyunghoon Koo,

Jaeyoung Lee, and Wook Hyun Kwon

Department of Control and Instrumentation Engineering

Seoul National University

ABSTRACT

In this paper, a high performance grahics system is suggested and its hardware architecture and software structure are described. The developed grahpics system is a multi-processing system that uses 6 i860 RISC CPU's and supports PHIGS language in a hardware level. The software is programmed with respect to the graphics pipeline and the software modules are distributed into each processor for the optimization of the performance. The implemented raphics system can draw about 100,000 3D polygons per second.

1. 서론

자동제어 분야가 발전되고 복잡해짐에 따라 실제 시스템에 대한 모의 실험이 중요시 되게 된다. 제어 시스템을 실제로 플랜트에 적용시키기 전에 그 동작 상태에 대한 충분한 모의 실험을 하여야만 위험 부담을 줄일 수 있게 된다. 과거의 시뮬레이션은 주로 수치해석에 따른 수치적 시뮬레이션이 주종을 이루고 있으나 컴퓨터의 그래픽 성능이 향상됨에 따라 점차로 수치적 결과 나열에서 실제 현상을 그대로 그래픽으로 표현하는 그래픽 시뮬레이션이 발전하는 추세에 있다. 이러한 추세는 공장 자동화의 대표적 기기인 로보트나 수치제어기기등에서 두드러지게 나타난다. 예를들면 로보트의 실시간 동작 모습등을 시뮬레이션하거나 전체 생산 라인의 동작 상태를 실시간으로 시뮬레이션하는 등의 작업등은 자동화 분야의 하나로서 중요한 역할을 하게 되었다. 그러나 이러한 자동화 시스템의 대부분은 고속의 반응 속도를 가지며 따라서 이들 자동화 시스템의 시뮬레이션을 위해서는 시뮬레이션 결과를 실시간으로 표현해주는 그래픽 시스템이 필수적으로 필요로 하게된다.

이러한 요구를 만족시키기 위해서는 고성능 고화질의 그래픽 시스템이 필요하게 되는데 원래 모습 그대로를 표현하기 위해서는 1280 X 1024 이상의 해상도를 가져야하며 색상도 16,000,000 가지를 가져야 한다. 또 실제 동작의 실감을 위해서 3차원으로 표시될 수 있는 모델링 기법이 필요하고 이 기법을 그래픽 시스템에서 지원해주는 것이 필요하게 된다. 실시간 화면 처리를 위해서는 고속의 그래픽 데이타 처리 속도를 요구 하게 됨으로 다중 프로세서를 이용한 고성능 시스템이 대두되고 있다.

본 연구에서는 위의 요구를 만족시키는 고성능 그래픽 시스템을 개발하는 것을 목적으로 하고 있다. 3차원 모델링을 위하여 표준 그래픽 라이브러리인 PHIGS를 사용하고 이를 하드웨어적으로 지원하여 그 처리속도를 높였다. 고속의 실시간 데이타 처리를 위해서 6개의 프로세서를 병렬 파이프라인 형식으로 연결하여 각 레벨간의 작업량을 조절하여 성능을 향상시켰다.

2. 3차원 그래픽

2.1 그래픽 파이프라인

컴퓨터 그래픽 처리 과정은 사용자가 기술한 그래픽 데이타가 실제로 화면에 표시되기까지의 일련의 순차적인 계산 과정이다. 이를 크게 다섯으로 나누면 아래와 같다.

(1) 그래픽 데이타 생성(generation)
그래픽 데이타를 생성하고 구조화하는 과정이다.

(2) 그래픽 구조 해석(traverse)
그래픽 데이타 구조를 해석하여 그래픽 처리의 기본 단위인 프리미티브(primivite)들을 생성하는 과정이다.

(3) 그래픽 데이타 변환(transformation)
세번째는 프리미티브에 대한 3차원 변환(transformation)및 광도 계산(lighting)을 하는 과정이다.

(4) 주사선 변환(scan conversion)
네번째는 계산 결과를 화면 메모리에 저장(rasterization)하는 단계로 주사선 변환(scanline conversion)이라고 한다.

(5) 표시(display)

마지막은 화면 메모리의 내용을 화면에 표시하는(display) 것이다.

초기의 컴퓨터 그래픽 시스템은 위의 다섯가지를 모두 하나의 프로세서(processor)가 처리하였으나 현재는 그래픽 서브시스템(graphics subsystem)이라는 전용 하드웨어를 따로 두어 위 과정중 (2) 이하 또는 (3) 이하의 과정을 처리하게 하고 있다. 그 이유는 첫째, 하나의 프로세서로는 모든 그래픽 작업들을 빠르게 수행할 수 없다는 것이고 둘째, 전용 프로세서에게 그래픽 작업을 맡김으로써 호스트는 운영체제 또는 사용자 연결, 응용 프로그램에 전념할 수 있게 되고 세째, 그래픽에 맞는 프로세서와 시스템의 구조가 가능해지기 때문이다. 그래픽 서브 시스템이 (1) 또는 (1),(2)를 포함하지 않는 이유는 (1),(2)의 과정은 모델링 및 그 해석에 관한 문제이므로 호스트내의 응용 프로그램이 처리하는 부분이기 때문이다.

그래픽 서브시스템의 구조는 파이프라인 식으로 여러 단계를 두어 각각 하나 또는 그 이상의 프로세서가 맡아서 처리하게 하고 있다. 이는 위에 설명한 각 과정을 프로세서 여러 개가 분담토록 한 것이다. 즉, 알고리즘(algorithm) 상의 그래픽 파이프라인(graphics pipeline)이 하드웨어 파이프라인으로 대체된 것이다(그림 1).

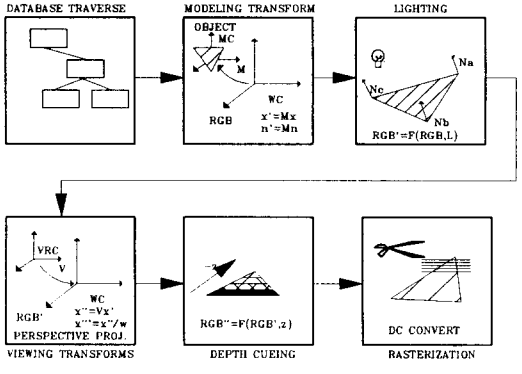

그림 1. 그래픽스 파이프라인

2.2 컴퓨터 그래픽 표준안

PHIGS(Programmer's Hierachical Interactive Graphics Sytem)는 1988년에 확립된 그래픽 표준안으로 스트럭쳐(Structure) 단위의 계층적 구조를 갖고 각 스트럭쳐는 스케일링(scaling), 회전, 이동, 조명에 따라 동적으로 변화할 수 있다. 이런 데이타베이스(database)의 변화는 자동적으로 화면의 갱신을 일으켜서 동적이고 복잡한 3차원 그래픽의 모델링이 가능하다. 이런 여러 장점들 때문에 현재 대부분의 그래픽 시스템에서 PHIGS, 또는 그와 유사한 구조의 계층적 그래픽 모델링 방법이 사용되고 있다. PHIGS는 더욱 발전되어 보다 실제적인 모델링을 위해 더 많은 프리미티브와 빛의 모델을 포함하는 PHIGS+가 1991년에 표준안으로 확정되었다. PEX(PHIGS Extentions to X window system)는 PHIGS의 개념을 X 윈도우 시스템에 적용시킨 것으로, 앞으로 네트웍(network)에 기초한 그래픽 시스템의 표준으로 사용될 전망이다.

PHIGS의 기능을 요약하면 아래와 같다. (PHIGS 용어는 그대로 사용하였다.)

- 각 스트럭쳐는 계층적인 구조를 이룬다.
- 구조 해석시에 바인딩(binding)이 된다.
- 네임 세트(name sets)와 필터(filters)를 가진다.
- 아래와 같은 좌표 변환 파이프라인을 가진다.

  모델링 좌표 변환

  관측 좌표 변환

  절단(clipping)

  워크스테이션 좌표 변환
- 스트럭쳐 수정 기능을 가진다.

본 연구에서는 PHIGS와 같은 계층적 구조의 동적인 그래픽 표준안을 지원할 수 있는 그래픽 시스템의 구조를 제시하고 이를 하드웨어로 구현하는 것을 목적으로 하고 있다.

3. 하드웨어의 구현

개발된 고성능 그래픽 시스템은 2열의 전체 병렬 파이프라인 구조를 가지고 있다. 이는 그래픽 프로세서 서브시스템, 그래픽 메모리 서브시스템, 화면 제어 서브시스템으로 나누어지고 그림 2와 같은 형태이다. 각 서브시스템은 독립된 보드로 구성하고 커넥터를 정의하여 접속하였다. 그래픽 서브시스템에 전달할 그래픽 구조를 생성하는 호스트도 함께 구현하였다.

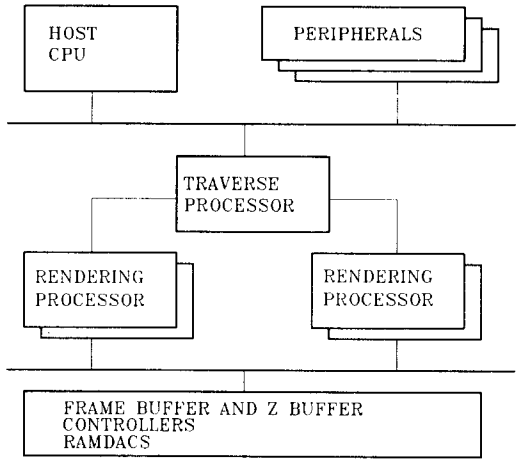

그림 2. 개발된 그래픽 시스템의 전체 구조

3.1 그래픽 프로세서 서브시스템

그래픽 프로세서 서브시스템은 한 개의 해석 프로세서와 네 개의 계산 프로세서로 이루어지고 네 개의 계산 프로세서는 2열로 전체 병렬화되어 있다. 모든 프로세서는 Intel의 64 비트 프로세서인 80860 프로세서를 사용하고 각 프로세서는 1 Mbyte의 내부 메모리를 가지고 있다. 이 시스템의 구조는 그림 3과 같다.

프로세서의 주변회로는 Xilinx의 FPGA를 써서 그 부피를 최소화하였고, 직렬 통신 포트는 시스템 개발 시에 터미날을 연결하거나 프로그램 다운로드를 위한 것이다. 보드의 크기를 줄이기 위해 이중 포트 메모리 모듈, SRAM 모듈, 직렬 통신 모듈등을 모듈화 하여 확장성과 유지 보수에 편리하게 하였다.

i860은 파이프라인 모드의 실수 계산이 가능하기 때문에 그래픽 처리 세번째인 실수 계산에 적합하고, 그래픽 전용 모듈

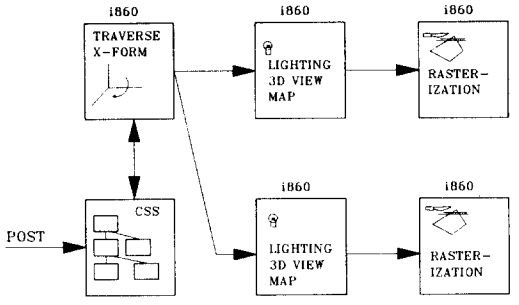

그림 3. 그래픽 프로세서 서브시스템

과 그래픽 명령을 가지고 있기 때문에 그래픽 처리 네번째인 주사선 변환에도 적합한 프로세서이다.

해석 프로세서와 2 열의 파이프라인은 두 개의 이중 포트 메모리로 연결되고 그래픽 명령의 전달은 해석 프로세서가 전담한다. 이중 포트 메모리는 256 KByte의 RAM으로 16 비트씩 전달한다. 파이프라인 내에서의 프로세서 연결도 이중 포트 메모리로 연결되고, 호스트와 해석 프로세서의 연결도 똑같은 방법을 쓴다.

그래픽 프로세서 서브시스템은 화소 버스(pixel bus)를 통해 그래픽 메모리 서브시스템과 연결된다.

3.2 그래픽 메모리 서브시스템

그래픽 메모리 서브 시스템은 화소 버스 제어기와 화면 버퍼로 이루어진다. 화면 버퍼는 많은 데이타 통신량에 대응하기 위해 다섯 뱅크로 나누고 5:1로 인터리빙하여 각 뱅크를 하나의 화소 버스 제어기가 담당한다. 이로 인해 메모리의 억세스 타임은 400 nsec 에서 80 nsec로 줄어 이를 역으로 환산하면 화소 버스의 통신폭은 12 Mpixel/sec 이다. 그래픽 메모리 서브시스템의 구조는 그림 4과 같다.

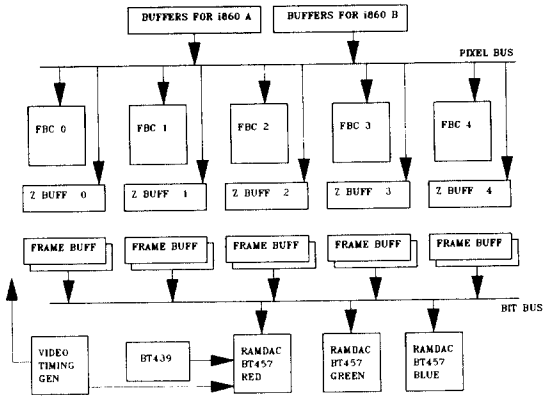

그림 4. 그래픽 메모리, 화면 제어 서브 시스템

3.2.1 화소 버스 제어기

화소 버스 제어기의 기능은 2 열의 그래픽 파이프라인에서 오는 데이타를 처리하기 위해 화소 버스를 시분할(time sharing)하여 각 파이프라인에게 사용하게 하는 것이다. 이로 인해 화소 버스는 버스 구조이지만 버스 조정이 필요없어지므로 화소 버스에 연결되는 두 개의 프로세서는 마치 내부 메모리에 데이타를 쓰는 것처럼 화면 버퍼를 사용할 수 있다.

화소 버스 제어기의 또 하나의 기능은 Z 버퍼 비교기로 화면 버퍼의 Z 값과 화소 버스를 통해 전송된 Z 값을 비교하여 전송된 Z 값이 클 때만 화면 버퍼를 갱신한다. 화소 버스 제어기는 Xilinx의 FPGA XC3064를 써서 설계하였다.

3.2.2 화면 버퍼

화면 버퍼는 RGB 각각 8 비트씩 24 비트를 쓰고, Z 버퍼는 24 비트를 쓰므로 1600만 가지의 색상 표현과 1600만 가지의 거리 표현이 가능하다. 화면의 화소수는 가로가 1280 화소, 세로가 1024 화소로 총 131만의 화소이다. RGB 버퍼는 두 개를 써서 한 RGB 버퍼에 데이타가 써지는 동안에는 다른 화면을 보여주게 하여 동적인 이미지 표현이 가능하다. 화면 버퍼는 9 x 1280 x 1024 = 11.8 MByte 의 메모리로 구성된다. 이 중 7.8 MByte의 RGB 버퍼는 화면 버퍼 전용의 VRAM이고, 4 MByte의 Z 버퍼는 DRAM으로 이루어진다.

3.3 화면 제어 서브시스템

화면 제어 서브 시스템(display controller subsystem)은 비디오 램이라 불리는 RGB 버퍼의 내용을 읽어서 모니터(monitor)에 전달하는 기능을 한다. 이 서브시스템의 특징은 동작 주파수가 매우 높다는 점과 디지탈 신호의 아나로그 변환이 일어난다는 점이다.

본 시스템의 화면은 1280x1024 화소의 고해상도와 화소당 24 비트의 실제 색상(real color)을 표시하고 비디오 램의 재생 시간(refresh time)을 없애기 위해 화면 주파수를 120MHz라는 고주파로 정하고 이에 맞는 클록 생성기(clock generator)를 사용하였다. 화면 메모리의 내용은 디지탈 값이고 모니터는 아나로그 값을 받아들이므로 중간에 변환기(D/A converter)가 필요하고 이 변환기에 아나로그 전원이 공급되므로 전원의 분리가 필요하다. 이런 특징들 때문에 화면 제어 서브 시스템에서는 소자의 배치, 아나로그 전원과 디지탈 전원의 분리, 노이즈의 제거 등에 대한 많은 고려를 하였다.였다.

4. 알고리듬 분석

4.1 모델링 및 구조 해석 알고리즘

사용자는 응용 프로그램을 통해 그리려는 대상을 모델링한다. 모델링시 사용되는 데이타의 기본 단위는 스트럭쳐 요소(structure element)이고 이 스트럭쳐 요소의 집합이 스트럭쳐이다. 응용 프로그램은 여러 스트럭쳐들을 모두 연결하여 계층 구조를 형성하고, 이를 CSS(Centralized Structure Store)라 불리는 메모리에 연결 리스트(linked list)의 형태로 저장한다. 이 계층 구조는 한 모델의 서로 다른 부분간의 논리적 관계를 정의하게 되고, 이러한 계층 구조는 한 모델에 여러 번 반복되는 데이타를 하나의 복사본만으로 모두 대체할 수 있다는 장점이 있다.

모델의 기하 정보(geometric information)와 속성(attribute)은 CSS에 각각 독립된 형태로 저장되고 그래픽 모델

을 해석할 때 표시할 부분(window), 시점(view point), 사용할 워크스테이션에 따라 같이 바인딩(binding)되어 지는데 이 과정을 트래버스(traverse)라고 한다. 이렇게 기하 정보와 속성을 따로 관리하면 동적인 그래픽을 구성할 때 데이타의 변경이 적어진다는 이점이 있다. CSS에 저장된 모델의 정보는 포스트(post)라는 명령에 의해 트래버스가 일어나고 화면에 그려지는데 그래픽 모델의 트래버스는 독립된 연결 리스트들의 바인딩 및 프리미티브의 생성, 그래픽 정보의 전달이다.

그래픽 정보를 3 개의 꼭지점을 갖는 3차원 도형(polygon)에 대한 것으로 가정한다면 트래버스 후에 생성되는 정보는 각 꼭지점(vertex)의 위치, 도형의 수직 벡터(normal vector), 칼라 정보이다. 이는 (x,y,z), (nx,ny,nz), (r,g,b)의 9 개의 실수 값으로 각 실수 값을 32 비트로 표시하면, 한 꼭지점마다 36 바이트가 필요하고 도형마다 108 바이트가 소요된다. 그래픽 서브시스템의 성능을 초 당 10만 도형으로 목표하면 트래버스 과정에서 필요한 통신폭(bandwidth)은 초 당 10 Mbyte 정도이다.

4.2 3차원 변환 알고리즘

사용자가 모델링 좌표(modelling coordinate) 또는 물체 좌표(object coordinate)로 정의한 그래픽 데이타는 일련의 좌표 변환 및 계산을 거쳐서 최종적으로는 화면 좌표(screen coordinate) 또는 기구 좌표(device coordinate)의 화소 단위로 출력되게 된다. 이 과정은 실수 데이타를 정수로 변환하는 과정을 포함한다. 3차원 변환 알고리즘에는 좌표계 변환 알고리즘, 광도 계산 알고리즘, 절단 알고리즘, 투상 알고리즘, 화면 좌표 변환 알고리즘 등이 있다.

4.2.1 좌표계 변환 알고리즘

3차원 좌표 변환에는 모델링 변환, 투상 변환의 두 가지가 있는데, 광원의 모델링을 간단히 할 경우에는 두 가지의 변환을 한 번의 행렬 곱셈으로 계산할 수 있다.

동차의 3차원 꼭지점은 (x,y,z,w)의 네 가지 성분을 포함하고 이의 변환은 이 좌표에 (4 x 4)의 행렬을 곱하는 것이다. 이 계산은 16 번의 실수 곱셈과 12 번의 실수 덧셈으로 이루어지는데, w 값을 1.0으로 고정시키면 실수 곱셈은 12 번으로 줄어든다. 이 계산량은 도형 당 72 회의 실수 계산이므로 초 당 10만 도형을 그리려면 7.2 Mflops의 성능이 필요하다.

수직 벡터는 w 성분이 필요하지 않으므로 그 변환은 좌표값에 (3 x 3)의 행렬을 곱하는 것이다. 이 계산은 9 번의 실수 곱셈과 6 번의 실수 덧셈으로 이루어진다. 이 계산량은 도형 당 45 회의 실수 계산이므로, 초 당 10만 도형을 그리려면 4.5 Mflops의 성능이 필요하다.

수직 벡터는 광도 계산에 필요한 벡터로 계산 전에 그 크기를 1로 정규화(normalization)해야 한다. 이 계산은 제곱의 합을 구할 때 3 회의 실수 곱셈과 2 회의 실수 덧셈이 필요하고, 제곱근의 역수를 구할 때 1 회의 실수 제곱근 계산과 1 회의 실수 역수 계산이 필요하고, 이 값을 각 성분에 곱할때 3 회의 실수 곱셈이 필요하다. 제곱근의 역수를 구하는 계산은 프로세서마다 다르지만 이 계산을 2 회의 실수 계산으로 환산하면 위의 계산량은 도형 당 30 회의 실수 계산이므로 초 당 10만 도형을 그리려면 3.0 Mflops의 성능이 필요하다.

정규화 계산이 벡터나 행렬곱에 비해 수행 시간이 긴 이유는 정규화 계산은 각 계산 단계가 상호 관련(dependent)되어서 벡타 모드 수행의 이점이 적어지기 때문이다.

4.2.2 광도 계산 알고리즘

광도 계산은 물체의 밝기와 색상을 계산하는 과정으로 빛의 모델링에 따라 전체 조명 모델(global illumination model)과 국부 조명 모델(local illumination model)의 두 가지 모델링 방법이 있다. 이 중 전체 조명 방법은 사진과 같은 섬세한 그래픽을 위한 것으로 광 추적법(ray tracing) 등이 있으나 고속 그래픽 처리에는 거리가 있으므로 본 연구에서는 도형의 꼭지점에 대해서만 광도 계산을 하는 국부 조명 모델을 분석한다.

모델링한 물체의 밝기 및 색상은 물체의 표면 특성, 광원의 종류, 광원과의 위치 관계, 관측자와의 위치 관계 등에 따라 계산되어진다. 광도 계산에 퐁(phong)의 모델을 사용할 경우 빛이 물체에 반사하여 눈이 지각하기까지를 간단히 계산하는데, 그것은 주변광(ambient light), 난반사(diffuse reflection), 정반사(specular reflection)이다. 광원의 종류에는 점 광원(positional light), 지향성 광원(directional light), 집중 광원(spot light)등이 있으나 본 연구에서는 무한대 위치의 방향을 가지고 빛의 감소가 없는 지향성 광원만으로 광도 계산을 분석한다. 또 관측자와의 위치 관계도 거리는 계산하지 않고 방향만 계산한 근사적인 계산을 한다.

벡터의 내적은 꼭지점 당 1 회만 계산하면 되며 3 회의 실수 곱셈과 2 회의 실수 덧셈이고 지수 계산은 표(lookup table)를 사용하면 되므로, 12 회의 실수 곱셈, 10 회의 덧셈, 3 회의 실수 비교, 1 회의 메모리 참조가 꼭지점의 광도 계산에 소요된다. 이는 도형 당 72 회의 실수 계산으로 초 당 10 만 도형을 그리려면 7.2 Mflops가 필요하다.

4.2.3 절단 알고리즘

3차원 도형은 Sutherland-Hodgman 알고리즘에 의해 절단된다. 이는 3차원 관측 공간(viewing volume)을 설정하고 벗어난 부분을 절단하는 것으로 이 알고리즘을 두 부분으로 나누면 절단을 위한 비교를 하는 부분과 실제 절단 처리를 하는 부분이다. 모든 도형은 절단을 위한 비교를 거치지만 실제 절단 처리가 되는 것은 경우마다 다르므로 절단을 위한 비교 부분만을 분석하면 꼭지점 당 6 회의 실수 비교가 필요하다. 이는 도형 당 18 회의 실수 비교에 해당하므로 초 당 10만 도형을 그리려면 1.8 Mflops의 성능이 필요하다.

4.2.4 투상 알고리즘

투상 알고리즘(projection algorithm)은 세계 좌표계로 주어진 도형의 좌표를 정규 투상 좌표계(normalized projection coordinate)로 변환하는 것으로 (4 x 4)의 관측 행렬 곱셈과 곱셈한 결과를 정규화하는 계산이다. 앞에 말한 바와 같이 광원 모델을 간단히 하였을 경우에는 이 중 행렬 계산을 생략할 수 있다. 그러나 도형의 좌표를 정규화하는 계산은 생략할 수 없고 이는 꼭지점의 x,y,z 값을 w로 나누는 것이다. 이 계산은 w의 역수를 구하는 실수 연산 1 회, 각 좌표를 w의 역수로 곱하는 실수

곱셈 3 회로 이루어진다. 이는 도형 당 12 회의 실수 연산이므로 초 당 10만 도형을 그리기 위해서는 1.2 Mflops가 필요하다.

4.2.5 화면 좌표 변환 알고리즘

정규 투상 좌표를 화면 좌표, 또는 기구 좌표로 변환하는 것으로 x,y,z 값을 화면의 주소에 알맞도록 바꾸어 주는 것이다. x,y의 계산은 화면의 해상도에 따라 달라지는데 1280 x 1024의 화면의 경우 x 값에는 1280, y 값에는 1024를 곱해야 한다. z의 계산은 Z 버퍼의 크기에 따라 달라지는데, 24 비트 Z 버퍼는 z 값에 2^{24}을 곱하면 된다.

z 값은 모델링 시에는 가장 가까운 거리를 0으로 정의하는데 설계의 편의상 Z 버퍼에 쓰느 값은 이를 바꾸어서 가장 먼 거리를 0으로 정의 하기도 한다. 이때는 z의 최대와 최소를 바꾸기 위해 -2^{24}을 곱하고 2^{24}을 더하면 된다. y 값의 최대, 최소가 바뀐 경우에도 z와 마찬가지로 실수 덧셈 1 회만 추가하면 된다.

이 계산은 3 회의 실수 곱셈과 2 회의 실수 덧셈으로 이는 도형 당 15 회의 실수 연산이므로, 초 당 10만 도형을 그리기 위해서는 1.5 Mflops 가 필요하다.

4.3 음영 처리 및 주사선 변환 알고리즘

음영 처리 알고리즘은 도형 내부의 색을 결정하는 알고리즘으로, 이에는 단색 음영 처리와 구로우 음영 처리 알고리즘, 퐁 음영처리 알고리즘 등이 있다. 단색 음영 처리는 도형의 내부점을 단색으로 결정하는 것이고, 구로우 음영 처리는 도형 내부점의 색상을 각 꼭지점 색상의 보간법으로 계산하는 것이고, 퐁 음영처리는 도형 내부점의 수직 벡터를 보간법으로 계산하고 각 점마다 광도 계산을 하는 것이다.

구로우 알고리즘으로 음영처리를 하는 과정은 아래와 같다.

(1) 볼록 다각형인지를 검사하고 아닌 경우 볼록 다각형의 조합으로 나눈다.
(2) 화면의 y 값에 따라 사다리꼴로 나눈다.
(3) 사다리꼴 좌우변의 기울기를 계산하고 좌우변의 색상을 보간법으로 구한다.
(4) 화면의 주사선에 따라 화면 버퍼에 도형의 색상을 채운다.

주사선 변환이란 (4)의 과정을 뜻하는 것으로, 모든 계산이 끝난 도형의 색 정보를 모니터의 전자총이 화면의 주사선을 진행하는 순서대로 화면 버퍼에 기록하는 것이다. 주사선 변환시 출력되는 값은 정수 값이므로 구로우 음영법은 주로 정수 계산에 의해 이루어 지는데 주의점은 오차가 없도록 기울기 계산시나 보간법 계산 시에 충분한 정확도를 유지해야 한다. 즉, 색 정보가 적색, 녹색, 청색 각각 8 비트 정수이지만 계산 시에는 소수점 이하 부분도 포함해서 각각 16 비트 이상의 정수로 계산을 해야한다.

주사선 변환의 성능은 한 주사선내에서 얼마나 빨리 계산하는가로 결정된다. 즉, 음영 처리 프로그램의 가장 안쪽 루프의 계산시간을 최대로 줄여야 하고 이 계산은 화소 어드레스의 증가, 각 색 정보의 증가, z 값의 증가, 색 정보와 z 값의 출력, 루프 계산으로 이루어진다. 이는 6 회의 정수 덧셈과 1 회의 메모리 억세스, 1 회의 분기로 화소당 모두 8 회의 정수 동작이므로 10 Mpixel/sec 의 성능을 위해서는 80 MOPS가 필요하다.

이를 i860으로 이중 명령 모드로 프로그래밍하고 Z 버퍼 비교기가 따로 있는 경우에는 화소 당 4 클록이 소요된다.

4.4 은면 처리 알고리즘

3차원 그래픽에서 그래픽 정보는 관측점으로 부터의 거리를 포함하고 그래픽 서브 시스템은 거리 정보에 따라 관측자와 가장 가까운 물체만을 표시해야 하는데, 이 과정을 은면 처리라고 한다. 은면 처리 알고리즘에는 주사선법(scan-line method), 깊이 정렬법(depth-sorting method) 등의 소프트웨어에 의한 알고리즘과 Z 버퍼 방법(Z buffer method)의 하드웨어에 의한 알고리즘이 있고 이중에서 고성능 그래픽 시스템에는 Z 버퍼 방법이 쓰인다.

Z-버퍼 방법은 화면을 저장하는 메모리에 화소의 색상을 나타내는 RGB 버퍼외에 거리를 저장하는 Z 버퍼를 두어 거리에 따라 화면 메모리에 쓸 것인지 아닌지를 결정하는 방법이다. 이 방법은 부가적인 메모리가 필요하지만 가장 빠른 은면 처리법이고, Z 버퍼를 비교하는 특수한 하드웨어가 추가되면 더욱 빠른 처리가 가능하다. 본 연구에서는 Z 버퍼 방법을 쓰고 Z 버퍼 비교 하드웨어를 포함한다. Z 버퍼 방법은 아래와 같은 단계로 요약된다.

(1) Z 버퍼와 RGB 버퍼를 초기화한다. 즉, Z 버퍼값은 최대 거리로 모두 채우고, RGB 버퍼값은 흑색으로 모두 채운다.
(2) 현재 쓰려는 화소의 RGBZ 값을 계산한다.
(3) 이미 쓰여진 화소의 Z 값을 읽는다.
(4) 현재 쓰려는 Z 값과 비교하여 현재 쓰려는 Z 값이 작을 경우에만 RGBZ 값을 버퍼에 기록한다.

위의 동작중 2 번을 제외한 부분은 화면 버퍼를 관리하는 특별히 설계된 하드웨어가 필요하고 초 당 10만 도형을 그리기 위해서는 10 Mpixel/sec의 메모리 억세스가 일어나므로 100 nsec 내에 3,4 번의 과정을 처리하도록 설계해야 한다.

도형 내부점의 z 값은 2.8 절의 구로우 음영법과 마찬가지로 도형의 꼭지점들의 z 값을 이용 보간법으로 구한다. 그러나 구로우 음영법과 다른 점은 도형은 한 평면 안에 있으므로 주사선 변환에 필요한 z 값의 증분은 도형 당 1 회만 계산하면 된다는 점이다.

4.5 알고리즘 수행의 균형화

위에서 계산한 알고리즘의 처리 시간을 표 1 에 요약하였다. 연산 회수는 한 도형에 필요한 실수 덧셈과 실수 곱셈의 회수이고 실 처리 시간은 이를 i860에서 수행하였을 때 걸리는 클록수이다.

이를 바탕으로 세 개의 i860 프로세서로 이루어진 그래픽 파이프라인을 가정하면 수행할 각 알고리즘들은 첫번째 프로세서에는 꼭지점 변환, 수직 벡터 변환, 두번째 프로세서에는 수직 벡터 정규화와 광도 계산, 마지막 프로세서에는 절단, 투상, 화면 좌표 변환을 할당한다.

알고리즘	연산 회수	실 처리 시간	비율
꼭지점 변환	72	60	19.4
수직 벡터 변환	45	39	12.6
수직 벡터 정규화	30	81	26.2
광도 계산	72	60	19.4
절단	24	24	7.8
투상	12	18	5.8
화면 좌표 변환	15	27	8.7
총 계	270 회	309 클록	100 %

표 1. 각 알고리즘의 처리 시간 및 비율

5. 결론

논문에서는 자동화 시스템의 실시간 시뮬레이션 등에 필요한 고속, 고화질의 그래픽 시스템의 개발 결과를 제시하였다. 개발된 그래픽 시스템은 고속의 RISC CPU i860을 6개 사용하여 병렬 파이프라인 구조를 갖는 멀티 프로세서 시스템으로 구성하였고 고속의 화소 처리를 위하여 화면 제어기는 6개의 전용 칩을 설계하여 제작 하였다. 개발된 시스템의 최종 성능은 Gauraud Shading을 하는 경우 100화소의 폴리곤을 초당 100,000개 그릴 수 있을 것으로 예산된다.

참고 문헌

[1] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics Principles and Practice*, Addison Wesley, pp.855 - 920, 1990.

[2] T. L. J. Howard, W. T. Hewitt, R. J. Hubbold, K. M. Wyrwas, *A Practical Introduction to PHIGS and PHIGS PLUS*, Addison Wesley, 1991.

[3] D. Shuey, T. P. Morrissey, "PHIGS: A Standard, Dynamic, Interactive Graphics Interface," *IEEE Computer Graphics & Applications*, August 1986.

[4] J. G. Torborg, "A Parallel Processor Architecture for Graphics Arithmetic Operations," *Computer Graphics (ACM)*, 21,4, July 1987.

[5] H. Niimi, Y. Imai, M. Murakami, S. Tomita, H. Hagiwara, "A Parallel Processor System for Three-Dimensional Color Graphics," *Computer Graphics (ACM)*, 18,3, July 1984.

[6] K. Akeley, T. Jermoluk, "High-Performance Polygon Rendering," *Computer Graphics (ACM)*, 22,4, August 1988.

[7] *Intel i860 64-bit Microprocessor Programmer's Reference Manual*, Intel, 1990.

[8] *Booktree Product Databook*, Brooktree, 1991.