# 모바일 3차원 그래픽 텍스처 매핑에 효율적인 새로운 유동형 고정 소수점 수 포맷

김남석°, 한정현
고려대학교 정보통신대학
{human°, jhan}@korea.ac.kr

# A new efficient format of dynamic fixed-point number for texture mapping in mobile 3D graphics

Namseok Kim°, Junghyun Han
College of information and communications , Korea University

**요 약**

본 논문에서는 텍스처 매핑을 처리하기 위한 텍스처 유닛 하드웨어 설계에 효율적인 새로운 유동형 소수점 포맷을 제안한다. 기존 고정 소수점 포맷은 하드웨어가 간단한 반면 고품질 텍스처 처리를 수행할 경우 오버플로우/언더플로우가 발생하며 부동 소수점 포맷은 이를 해결할 수 있으나 하드웨어가 복잡하다. 제안한 방식은 오버플로우/언더플로우를 해결하면서 부동소수점보다 하드웨어 크기를 줄여서 본 포맷을 적용한 가산기는 부동소수점보다 26% 작으며 곱셈기는 고정/부동 소수점보다 절반 이상으로 작다. 따라서 제안한 포맷은 100Mhz 이상의 빠른 동작이 가능하며 모바일 3차원 그래픽 가속기의 텍스처 유닛 설계에 효과적이다.

## 1. 서론

3차원 그래픽은 초기에 성능 제약으로 영화 특수 효과, 과학/의료 영상 등의 비 실시간 렌더링에 국한되었으나 전용 고속 하드웨어 가속기의 발전에 힘입어 실시간 렌더링이 요구되는 게임, 가상 현실 등에 빠르게 부응함으로써 멀티미디어 중 가장 주목 받는 분야가 되었다. 최근에는 PC나 아케이드 게임 또는 콘솔 게임 기반뿐만 아니라 PDA, 휴대용 게임기, 이동 전화와 같은 휴대형 기기에서도 3차원 그래픽이 주목 받고 있다. 그러나 3차원 그래픽은 고성능을 요구하는 반면 휴대형 기기는 칩 크기, 전력 소모 측면에서 PC등의 기기보다 제약이 크기 때문에 더욱 우수한 가격대 성능비를 보장하는 최적화된 아키텍처를 요구한다.

3차원 그래픽 가속기는 크게 기하학 연산 처리기(Geometry processor)와 렌더링 처리기(Rasterizer)로 이루어진다. 기하학 연산 처리기는 렌더링 할 객체를 구성하는 폴리곤의 크기/위치 변화 및 그에 따른 빛의 변화를 처리하는 부분(그림 1)이고 렌더링 처리기는 폴리곤을 픽셀로 변환하여 렌더링 하는 부분이다. 렌더링은 폴리곤 영역을 채울 내부 색상을 결정하며 텍스처 매핑, 깊이 비교 등 여러 과정(그림 2)을 거친다[1].

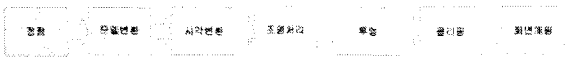

그림 1) 기하단계 파이프라인

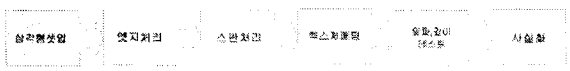

그림 2) 렌더링 처리 단계

이중 텍스처 매핑은 폴리곤 영역을 사용자가 지정한 외부 이미지로 렌더링 하는 작업으로 객체를 현실감 있게 표현하기 위해 매우 중요한 3차원 그래픽의 핵심 기술이다.

텍스처 매핑은 매 정점마다 지정된 텍스처 좌표에 의해 폴리곤에 씌울 이미지 영역을 결정하고 필터링을 통하여 새로운 픽셀을 생성(그림 3)하게 된다[2]. 그러나 실제로는 폴리곤과 이미지 사이의 크기 불일치로 여러 가지 왜곡이 발생하기 때문에 매핑 과정에서 보정을 위한 추가적인 계산이 필요하다.

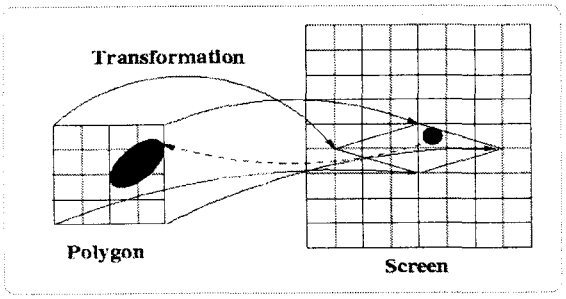


그림 3) 텍스처 매핑 과정

3차원 그래픽 가속기는 기하학 연산 처리기의 경우 주로 부동소수점 포맷을 사용하고, 렌더링 처리기의 경우 고정소수점 포맷을 사용한다. 렌더링 처리기는 기하학 연산 처리기에 비해 수치의 동적 범위가 훨씬 작기 때문에 하드웨어 비용이 적은 고정소수점 포맷으로 충분히 구현 가능하다.

그러나 텍스처 매핑은 일반적인 렌더링 처리보다 훨씬 넓은 수의 범위를 필요로 하므로 기존의 고정소수점 포맷으로는 언더플로우/오버플로우 문제를 해결하기 힘들다. 부동소수점의 경우, 이러한 언더플로우/오버플로우 문제를 해결할 수 있으나 하드웨어 비용이 너무 크다는 단점이 존재한다. 따라서 본 논문

은 텍스처 매핑에 효율적인 새로운 고정 소수점 포맷을 제안한다. 새로운 포맷은 고정소수점 포맷보다 훨씬 넓은 수 영역 범위 표기가 가능하면서 하드웨어 비용은 부동소수점에 비해 매우 작은 장점을 지닌다.

본 논문은 제 2장에서는 텍스처 매핑 기존 포맷들의 문제점과 새 포맷의 필요성을 분석하고 3장에서는 본 논문이 제안한 포맷에 대해 기술한다. 4장에서는 제안한 포맷을 이용하여 구현한 텍스처 유닛의 하드웨어를 비교 분석하고 5장에서 결론을 맺는다.

## 2. 텍스처 매핑

텍스처 매핑 시 월드 상에서 직교 투영을 적용할 경우 한 픽셀에 해당하는 공간상의 세그먼트의 길이가 같지만 원근 투영일 경우에는 z값에 의해 길이가 변하기 때문에 정점 속성들을 적절한 알고리즘을 사용하여 보간 해야 한다[3][4]. 이것을 원근 보정 또는 원근 교정(perspective correct)이라 하는데 이를 무시하면 기울기 불일치 라는 현상이 발생하여 원근감 없이 단순 선형 보간된 텍스처가 표현된다. 이것은 많은 메모리를 요구하기 때문에 PC 등과 같이 큰 제약이 없는 장치에 적용되는 것이었으나 사용자의 수준이 높아지고 기술이 발전함에 따라 많은 제약 조건이 따르는 휴대형 기기에서도 적용하게 되었다.

또한 이미지가 적용되는 방식 중 처리 방법을 결정하는 것을 OpenGL에서는 래핑 모드(Wrapping Mode)라 하고 Direct3D 는 텍스처 주소 지정 모드(Texture Addressing Mode)라 한다. 이 모드의 종류에는 랩(wrap) 또는 반복 (repeat), 거울상(mirror), 고정(clamp) 그리고 경계(border) 가 있다[1].

OpenGL-ES[5] 에서는 16.16 포맷(그림 4)의 고정소수점 사용을 권장하지만 그 이외에 처리 과정에 따라 오버플로우 및 언더플로우를 방지하기 위하여 8.24 및 24.8 등을 혼용한다. 표 1은 이러한 3가지 포맷의 표현 범위와 정밀도를 나타낸 것이다. 그러나 원근 텍스처링은 고정소수점을 이용하여 보간을 할 경우 언더플로우를 방지하기 위하여 정수부와 소수부는 각각 2.30 정도 되야 한다. 그러나 이 경우 정수부가 너무 작기 때문에 래핑 모드를 적용할 경우 오버플로우가 발생하게 된다.

메사[6]와 같은 소프트웨어 기반 렌더러일 경우 이러한 문제를 해결하기 위하여 부동 소수점[7]과 고정 소수점을 혼용하여 사용하고 있으나 하드웨어 가속기로 구현 하기에는 연산기의 크기 증대로 인해 적합하지 않다.


그림 4) 16.16고정 소수점 포맷

| 포맷 | 정수 표현 범위 | 소수 정밀도 |
|---|---|---|
| 16.16 | -2^15 ~ +2^15 - 1 | 1 / 2^16 |
| 8.24 | -2^7 ~ +2^7 - 1 | 1 / 2^24 |
| 24.8 | -2^23 ~ +2^23 - 1 | 1 / 2^8 |

표 1) 고정 소수점 표현 범위와 정밀도

따라서 기존의 고정 소수점보다 표현할 수 있는 범위가 정수부/소수부 모두 충분히 크면서도 부동 소수점보다는 훨씬 회로가 간단한 새로운 포맷이 필요하다.

새로이 제안하는 포맷은 값을 표기하는 범위가 32비트 고정 소수점에 비해 충분히 넓은 반면 텍스처 매핑이 정밀도에 덜 민감하다는 것을 감안하여 정밀도를 대폭 낮춤으로 효율적인 구조를 띄게끔 설계하였다. 따라서 부동소수점보다 훨씬 간단한 하드웨어로 고정소수점 연산에서 야기됐던 오버플로우나 언더플로우 문제를 해결하였다. 제안한 포맷은 효율적인 구조의 텍스처 유닛 하드웨어 설계를 가능하게 하므로 모바일 3차원 그래픽 가속기에 매우 유용하다.

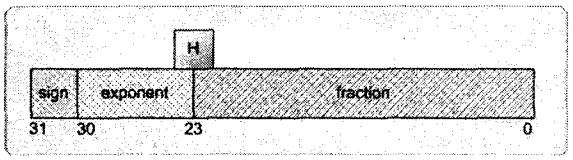


그림 5) 단일 정밀도 부동 소수점 포맷

## 3. 제안한 유동형 고정 소수점 포맷

### 3.1 유동형 고정 소수점 포맷

본 논문에서 제안하는 총 24비트 체계를 사용한 유동형 고정 소수점 포맷은 그림 6 과 같은 형태를 띄게 된다.

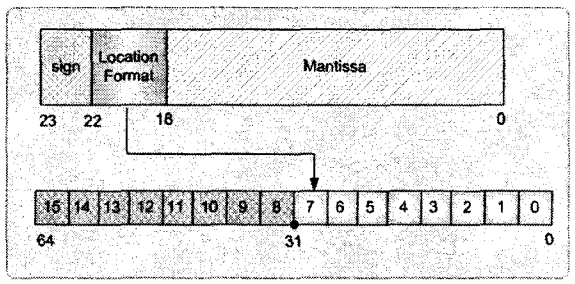


그림 6) 유동형 고정 소수점 포맷

이는 총 24비트의 데이터 패스로 구성되며 1비트의 부호(sign), 4비트의 위치 포맷(Location Format) 그리고 19비트의 가수(Mantissa)를 갖는다.

부호는 부동 소수점 및 고정 소수점과 마찬가지로 0이면 양수 1이면 음수를 나타낸다. 4비트의 위치 포맷은 4비트단위로 가수부분의 시작 위치를 나타내며 그 값은 가수부분의 위치 값이 된다. 그림에서는 7을 가리키고 있으므로 위치포맷의 값은 0x0111이 되고 나머지 19비트는 가수를 표현한다.

그림의 하단부분의 64비트는 가수를 표현한 것인데 상위 8개의 수(digit)는 정수 부분을 나타내며 하위 8개는 소수를 의미하게 된다. 즉, 항상 7부터 소수점이 시작하게 된다. 예를 들어 위치포맷의 값이 0x0000일 경우 소수 정밀도가 (1/2) ^ (32+15) 만큼 정밀도를 갖는다.

이러한 구조는 4비트(1 digit)단위로 움직이고 그만큼 검색 속도가 빠르기 때문에 순차적으로 값을 검사(reading on detection) 할 때에 속도상의 많은 이점이 있게 된다.

그림 7은 0x36000F가 유동형 고정 소수점으로의 표현이 어떻게 되는지 도식화 한 것이다.

0x36000F 는 2진수로 0011 0110 0000 0000 0000 1111 이며 이를 24비트 포맷에 정렬하면 0 0110 1100 0000 0000 0001 111x 이다.

그러므로 위치 포맷은 6 이 되며 소수점은 위치포맷 7 부터 시작하기 때문에 6이 가리키는 첫 번째 비트는 2/1^5 이 되는 것이다.

또한, 유동형 고정 소수점을 표현한 수식은 그림 8과 같다. 부호 비트가 0일 때는 양수, 1일 때는 음수를 나타내는 것을 수식으로 표현한 것이 (-1) ^ s 이다. 0.f 는 소수점 표현을 의미하며 4^l-7 은 4비트를 기준으로 위치포맷에서 7을 빼줘야 함을 의미한다.

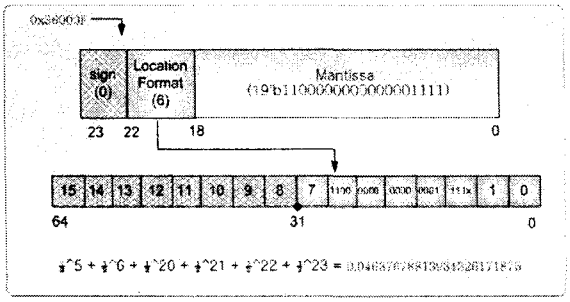


그림 7) 유동형 고정 소수점 표현의 예

$$F = (-1)^{s} * (2^{(e-127)}) * 1.m$$

그림 8) 유동형 고정 소수점 수식

### 3.2 유동형 고정 소수점 연산

그림 9는 제안하는 유동형 고정 소수점 연산의 곱셈을 나타내는 가상 코드이다.

두 입력의 부호를 배타적 논리합(XOR)하면 결과의 부호가 되며 각 가수를 곱한 후 최상위 4비트의 값이 0일 경우 포맷의 일치를 위해 가수를 4비트 쉬프트 후 상위 19비트의 값을 취하게 되며 보정을 위해 위치 포맷에 1을 빼주게 된다.

최상위 4비트만 검사하는 이유는 가수의 최상위 수(digit)엔 적어도 1이상의 값을 갖기 때문이다.

예로 들면 가수가 0001xx * 0001xx으로 시작한다고 가정했을 때 서로 곱한 결과는 0000 0001 xx가 되어 최상위 비트는 0000 이므로 포맷에서 1을 추가로 빼줘야 한다.

고정 소수점의 경우 32(비트) * 32(비트)를 해야 하지만 이 포맷의 경우 19(비트) * 19(비트)를 한 후 간단한 보정만 하면 되므로 크기는 줄고 속도는 빨라지게 된다.

나눗셈의 경우 A/B의 식이 있다면 A * 1/B 과 같은 의미를 갖기 때문에 곱셈 알고리즘과 방식이 같게 된다.

덧셈과 뺄셈의 경우 두 부호가 다를 경우 2의 보수를 취한 후 위치 포맷을 참조 하여 정규화(alignment)를 하게 된다. 여기서 정규화라 함은 작은 연산자의 지수를 증가시킴과 동시에 가수를 그 연산자의 지수가 다른 연산자의 주시와 같아질 때까지 이동시켜 가수를 배치하는 것을 말한다.

부동소수점의 경우 정규화를 위해 지수를 참조하여 -127((-2^7 - 1)에서 128(2^7)까지 쉬프트 연산(배럴 쉬프터가 당당)이 필요하게 되는데, 이는 그 규모가 크고 속도의 저하를 야기시키게 된다. 그러나 제안하는 구조에서는 정규화를 간단히 5개의 인덱스를 갖는 케이스 문(MUX)으로만 구현하기 때문에 1비트의 연산만을 필요로 하므로 절차가 매우 간단해진다.

정규화를 한 후 실제 덧셈 또는 뺄셈 연산이 이루어지고 그 결과 순차적인 값의 검사(reading on detection)를 하게 되는데 부동소수점 연산에서는 비트단위로 이루어 져야 하는 반면 제안한 유동형 고정 소수점에서는 4비트(digit)단위로 이루어 지기 때문에 검사가 빠르고 가볍게 된다.

그림 10은 덧셈 과정을 나타내는 순서도 이다.

```
sign = sign1 ^ sign2
tmp_mantissa = mantissa1 * mantissa2
tmp_lf = lf1 + lf2 - 7

lf ( (tmp_mantissa최상위 4bit) == 0 )
{
  mantissa = tmp_mantissa << 4
  location_format = tmp_lf - 1
}
else
{
  mantissa = tmp_mantissa
  location_format = tmp_lf
}
```

그림 9) 유동형 고정 소수점의 Pseudo Code

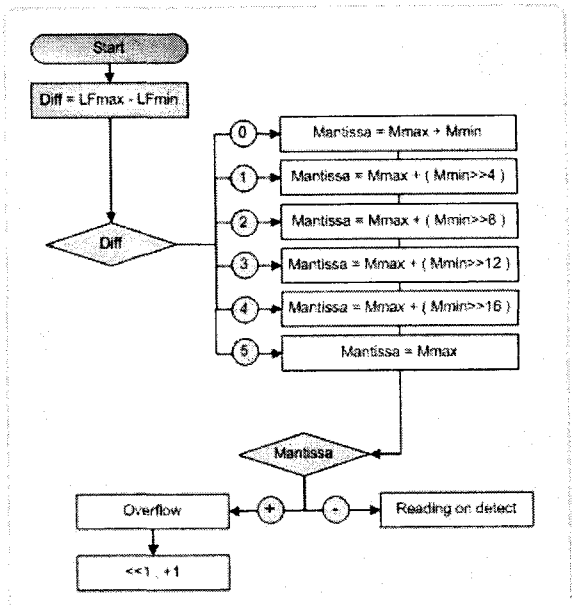


그림 10) 유동형 고정 소수점의 덧셈 과정

최초 각 위치 포맷의 값을 비교 후 차를 구한 후(Diff) 그에 따른 가수 값(Mantissa)을 구한다. 이 후 가수 값의 부호에 따라 양수이면 오버플로우가 발생하기 때문에 좌로 쉬프트 한 후 1을 더하고 음수이면 순차적인 검사 과정을 통하여 값을 정하게 된다.

### 4. 성능 평가

본 논문에서는 텍스처 유닛을 설계하는데 가장 빈번하게 사용되는 데이터패쓰인 가산기와 곱셈기에 대하여 제안한 고정 소수점 포맷, 16.16 고정 소수점 포맷, 부동 소수점 포맷을 기반으로 회로를 구현하여 각각 비교하였다(그림11).

구현한 가산기와 곱셈기는 0.13um 공정 라이브러리를 사용할 경우 각각 최고 50Mhz에서 단일 클럭으로 수행이 가능하다.
표 2를 보면 가산기의 경우, 제안한 유동형 고정 소수점 포맷 기반의 회로는 총 1209 게이트로 구성되어 부동형 소수점 포맷 가산기보다 26% 가량 작다. 16.16 고정 소수점 가산기보다 크지만 전체 텍스처 유닛을 구성하는데 있어서 언더플로우/오 버플로우를 완벽히 해결할 수 있다는 사실을 감안하면 허용 가능한 수치이다.
곱셈기의 경우, 제안한 포맷의 곱셈기는 총 3012 게이트로 고정 소수점 포맷이나 부동 소수점 포맷에 비해 절반 이하로 월등히 작음을 알 수 있다. 따라서 가산기/곱셈기 두 가지 결과를 종합해 볼 때 제안한 포맷은 원근 텍스처 처리 등 고품질 텍스처 매핑 보정 시 야기되는 오버플로우/언더플로우 문제를 완벽히 해결하면서도 작은 하드웨어 사이즈로 구현이 가능하다. 실제로 100Mhz 이상의 클럭 주파수에서 단일 클럭 사이클로 가산기와 곱셈기를 동작시키려고 하면 기존 고정 소수점 포맷은 곱셈기에서, 부동 소수점 포맷은 가산기에서 제약을 받는다. 이는 표현 가능한 수가 너무 세밀하기 때문이다. 반면 제안한 유동형 고정 소수점 포맷은 표현 가능한 수가 텍스처 매핑이 요구하는 적당한 수준으로 덜 세밀하면서도 충분히 큰 범위를 수용하므로 상대적으로 하드웨어 사이즈가 작고, 따라서 충분히 100Hz 이상에서 동작이 가능하다.

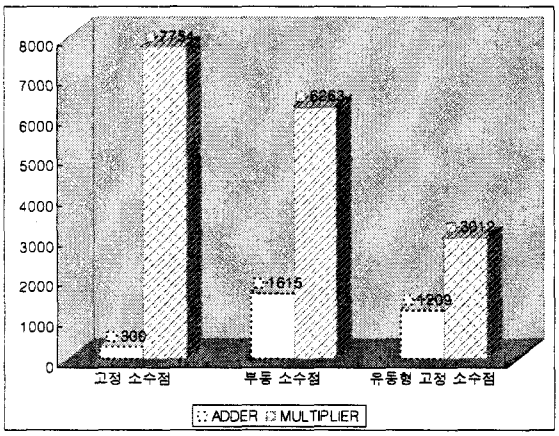


그림 11) 각 포맷의 연산자 크기 그래프

| | ADDER | MULTIPLIER |
|---|---|---|
| 고정 소수점 | 300 Gate | 7754 Gate |
| 부동 소수점 | 1615 Gate | 6263 Gate |
| 유동형 고정 소수점 | 1209 Gate | 3012 Gate |

표 2) 각 포맷의 연산자 크기

그림 9는 8.24 고정 소수점 연산 기반 하의 텍스처 매핑 영상과 제안한 유동형 고정 소수점의 영상을 비교한 것이다. 8.24 고정 소수점 포맷을 사용한 좌측 그림은 고정소수점 연산의 사용시 앞서 언급했던 언더플로우 현상으로 값의 오차에 의해 이미지가 사선으로 왜곡 되는 현상이 발생하나 제안한 포맷을 사용한 우측 그림은 언더플로우 영향 없이 정상적인 결과가 출력됨을 알 수 있다.

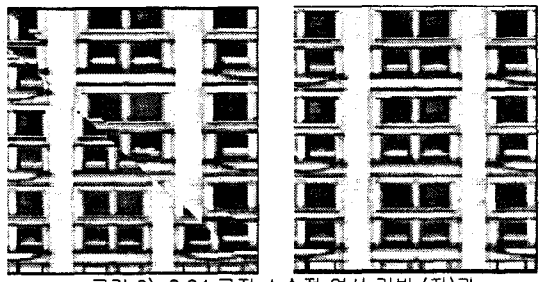

그림 9) 8.24 고정 소수점 연산 기반 (좌)과 유동형 고정 소수점 연산 기반 (우)의 원근 텍스처

## 5. 결론

본 논문에서는 텍스처 매핑을 처리하기 위한 텍스처 유닛 하드웨어 설계에 매우 효율적인 새로운 형태의 유동형 소수점 포맷을 제안하였다. 기존의 고정 소수점 포맷은 하드웨어가 간단한 반면 원근 텍스처링 등 고품질 텍스처 처리를 수행할 경우 오버플로우/언더플로우 문제가 발생한다.
부동 소수점 포맷은 이런 오버플로우/언더플로우 문제를 해결할 수 있으나 하드웨어 복잡도가 매우 크다는 단점이 존재한다.
제안한 방식은 고정 소수점 포맷보다 훨씬 넓은 수의 범위를 지원함으로써 오버플로우/언더플로우 문제를 해결하면서 부동소수점보다 표현할 수 있는 수의 세밀도를 낮춤으로써 하드웨어 크기를 줄였다. 따라서 이와 같은 방식을 적용할 경우, 100Mhz 이상의 빠른 동작이 가능하다. 이처럼 제안한 포맷은 높은 하드웨어 대 성능비를 보이므로 하드웨어 제약이 심한 모바일 3차원 그래픽 가속기의 텍스처 유닛 설계에 효과적인 적용이 가능하다.

## 6. 참고문헌

[1] Tomas Moller and Eric Haines, " Real-Time Rendering," pp.22-33, July 1988.
[2] Frederick M. Weinhaus, " Texture Mapping 3D Models of Real-World Scenes," ACM Computing Surveys, Vol. 29, No. 4, December 1997.
[3] B. Barenbrug, et al., " Algorithms for Division Free Perspective Correct Rendering" Proceeding of SIGGRAPH /EUROGRAPHICS workshop on Graphics Hardware, pp. 7-13, 2000.
[4] Donghyun Kim; Lee-Sup Kim " Division-Free Rasterizer for Perspective-Correct Texture Filtering" Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on Volume 2, 23-26, May 2004
[5] OpenGL ES, http://www.khronos.org/opengles
[6] The Mesa 3D Graphics Library, http://www.mesa3d.org
[7] ANSI/IEEE Std 754-1985 "IEEE standard for binary floating-point arithmetic" IEEE, 1985