4. 3차원 컴퓨터그래픽스

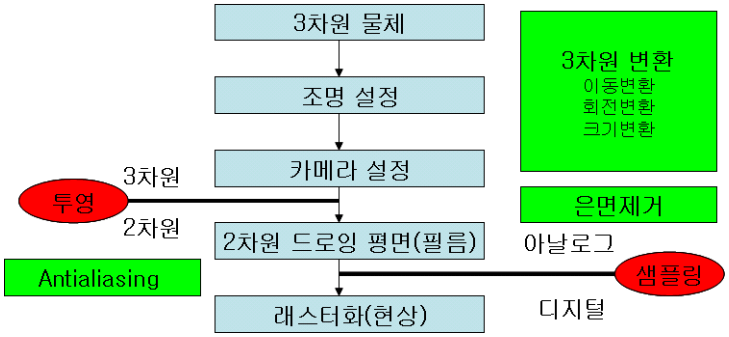


Fig 4.1 3차원 컴퓨터그래픽스 과정

3차원 컴퓨터그래픽스는 가상의 공간을 만들어 카메라로 찍은 장면을 컴퓨터 화면에 이미지나 동영상으로 보이게 하는 기술이다. 우선 3차원 공간에 물체를 만들어서 배치하며 원하는 색상을 가지는 조명을 원하는 위치에 배치시켜 스위치를 켜면 물체를 볼 수 있게 되는데 원하는 결과 이미지를 만들기 위해서 원하는 형태의 렌즈를 부착하여 원하는 위치, 원하는 방향으로 카메라를 위치시켜 원하는 형태로 투영한다. 이와 같은 물체, 조명, 카메라를 설정하는 과정은 3차원 공간에서 이루어지는 작업들이다. 3차원 물체를 카메라로 찍어 그 영상을 필름에 담는 과정인 투영 작업부터는 작업 공간이 2차원으로 바뀌게 된다. 이러한 투영작업에는 보이지 않는 면은 그리지 않도록 하는 은면제거법(Hidden-Surface Removal)이 필요하다. 투영하는 방법에는 직교투영과 투시투영이 있는데 보통 우리 사람 눈이 물체를 볼 때는 투시투영을 사용하고 있으며, 3D CAD와 같은 응용에서는 길이를 잘 유지하기 위하여 직교투영을 사용하기도 한다. 이렇게 투영된 이미지는 아날로그 형태이며 점과 직선으로 이루어져 있다고 보면 된다. 우리가 이미지를 나타내려고(현상) 하는 곳이 컴퓨터 모니터이기 때문에 픽셀 형태의 이산화 하는 작업(래스터화)을 거쳐야 한다. 아주 매끄러운 직선은 여러 개의 픽셀로 표현하기 때문에 계단 같은 모양으로 변형되는 셈이다. 이렇게 래스터화된 이미지에 대한 정보는 프레임버퍼라는 메모리에 0과 1로 저장된다. 저장된 값이 1 이면 그 픽셀은 그려지는 것, 저장된 값이 0 이면 그 픽셀은 빈 공간으로 처리하라는 의미가 된다. 만약 색상을 사용한다면 여러 장의 프레임 버퍼를 사용하여야 한다. 즉 담을 수 있는 공간이 필요한 셈이다. 컴퓨터 그래픽스에서는 빨강, 초록, 파랑을 기본 색상으로 사용하는데, 빨강을 8등분, 초록을 8 등분, 파랑을 8 등분하여 표현하려고 한다면, 빨강을 담기 위해서는 3($2^3 = 8$)장의 프레임버퍼, 초록을 담기 위해서는 3($2^3 = 8$)장의 프레임버퍼, 파랑을 담기 위해서는 3($2^3 = 8$)장의 프레임버퍼, 총 9장의 프레임버퍼를 사용하여 $512 = 8 \times 8 \times 8$ 개의 색상을 담을 준비를 하게 되는 셈이다. 그 다음 해야 할 작업은 필름에 담긴 이미지를 원하는 응용윈도우 어디에 어떤 크기로 보이게 할지를 정해야 한다. 즉 사진을 현상할 때, 3x4 형태 사진인지 4x6 형태 사진인지 정하듯이 컴퓨터그래픽스에서는 Viewport를 지정해 줘야 한다. 그러면 프레임버퍼에 담겨져 있던 디지털 정보가 디지털-아날로그 변환기를 거쳐 CRT의 전자총에 전달되어 전자를 쏘게 된다. 전자는 상하/좌우 편향장치를 거치면서 화면의 원하는 위치로 날아가서 브라운관 표면에 부딪히게 되면서 표면에 발려져 있는 형광물질과 작용하여 빛을 발하게 된다. 발하는 빛은 형광물질의 종류에 따라서 발하는 시간이 정해져 있는데 컴퓨터 화면에서 원하는 이미지를 계속 보게 하기 위해서는 동일한 지점에 전자를 계속해서 제공해줘야 한다. 보통 60Hz 모니터라는 것은 1초에 60번을 쏜다는 의미이다.

4.1 물체 모델링

대부분의 3차원 물체는 속과 겉이 구분되는 형태로 구성되며 이를 효율적으로 표현하기 위해서는 겉면만을 표현하는 Boundary 표현법을 사용한다. 내부는 보이지 않기 때문에 표면을 어떻게 표현하느냐에 많은 관심을 가지게 된다. 예를 들면, 책과 같은 형태는 6개의 면으로 이루어진 육면체 형태이며 따라서 이를 표현하기 위해서는 8개의 정점의 좌표가 필요할 것이며 6개의 면은 이러한 8개의 정점 중에서 4개씩 이루어진 형태를 이루고 있다. 여기에서 책과 같은 느낌을 주기 위해서는 표면에 원하는 색상과 모양을 띠는 이미지를 올려놓게(Texture Mapping) 되면 책과 같은 느낌을 받을 수 있게 된다. 이와 같이 대부분의 3차원 물체들은 가공된 물체들이며 이들은 정점들의 위치와 면을 형성하는 정점들의 리스트로 표현하는 방식을 사용한다. 이렇게 구성된 정점들과 선분들은 메쉬 라고 불리는 자료 형태로 저장된다 (Fig 3.2).

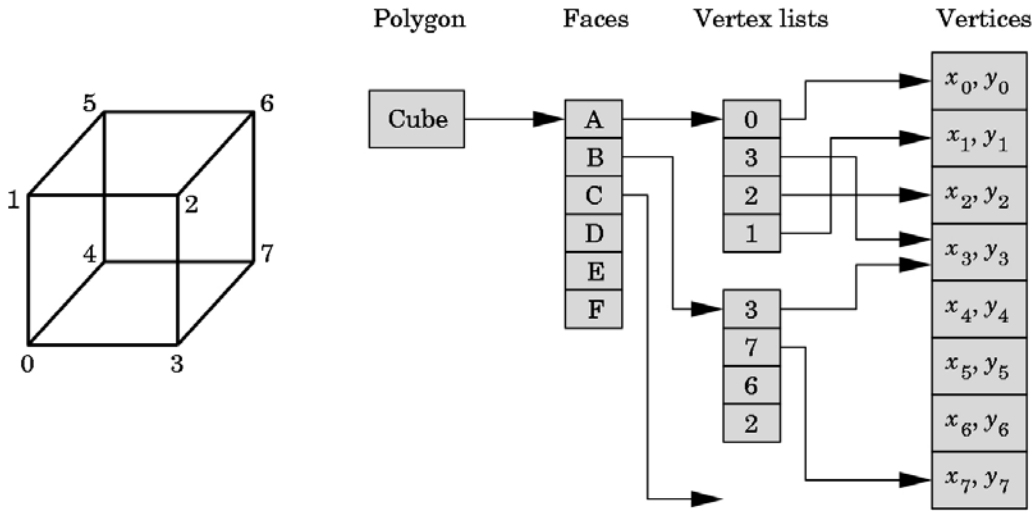


Fig 4.2 육면체 자료 구조

이러한 자료구조를 사용하기 때문에 메쉬를 저장할 때 효과적으로 처리하기 위하여 다음과 같은 indexed mesh description 방법을 사용하여 cube.txt 파일에 저장한다.

```
 8              // 정점의 수
2.0 0.0 0.0     // V[0] 의 좌표 x, y, z
2.0 0.0 2.0     // V[1] 의 좌표 x, y, z
2.0 2.0 2.0     // V[2] 의 좌표 x, y, z
2.0 2.0 0.0     // V[3] 의 좌표 x, y, z
0.0 0.0 0.0     // V[4] 의 좌표 x, y, z
0.0 0.0 2.0     // V[5] 의 좌표 x, y, z
0.0 2.0 2.0     // V[6] 의 좌표 x, y, z
0.0 2.0 0.0     // V[7] 의 좌표 x, y, z
6               // 면의 수
0 3 2 1         // F[0] 의 정점 번호 4개 (사각형)
2 3 7 6         // F[1] 의 정점 번호 4개 (사각형)
6 7 4 5         // F[2] 의 정점 번호 4개 (사각형)
5 4 0 1         // F[3] 의 정점 번호 4개 (사각형)
1 2 6 5         // F[4] 의 정점 번호 4개 (사각형)
0 4 7 3         // F[5] 의 정점 번호 4개 (사각형)
```

Fig 4.3 Indexed mesh description

이와 같이 저장된 육면체 자료를 응용프로그램에서 Load 하기 위해서는 다음과 같은 코드가 필요하다.

```
FILE *fp1;
char *filename;
struct point3d{ double x, y, z; };
point3d      *V;
struct face{ int v1, v2, v3, v4; };
face         *F;

filename = "cube.txt";
if ( (fp1 = fopen(filename, "r")) == NULL ) {
     cout << "  Can not compute \n";
     exit(1);
}
fscanf(fp1,"%d",&v_no); //input vertex number
V = new vertexdata[v_no];
for (int i = 0; i < v_no; i++)
     fscanf(fp1,"%f %f %f", &(V[i].x),&(V[i].y),&(V[i].z));

fscanf(fp1,"%d",&f_no); //input faces number
F = new face[f_no];
for ( i = 0; i < f_no; i++ )
     fscanf(fp1,"%d %d %d %d", &(F[i].v1), &(F[i].v2), &(F[i].v3), &(F[i].v4));
```

Fig 4.4 Indexed mesh description의 Data를 읽어 들이는 코드

그렇다면 이와 같은 사각형을 나타내는 방법은 어떨까 한번 간단하게 살펴보면 다음과 같다. 첫 번째 사각형인 F[0]은 V[0], V[3], V[2], V[1]의 네 개의 정점으로 구성되어 있다. 그러므로 네 개의 정점을 나타내는 glVertex3f 함수를 사용하고 네 개의 점으로 구성되는 도형이 다각형이기 때문에 도형의 타입을 POLYGON으로 설정한다. 만약 타입을 GL_POINTS 로 한다면 네 개의 점만 있는 그림을 볼 수 있게 된다. 육면체를 완성하기 위해서는 이러한 코드를 6개 면에 대해서 모두 작성되어야 함을 알 수 있다.

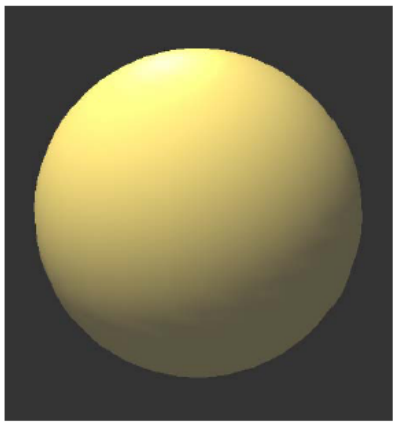
```
glBegin(GL_POLYGON);
   glVertex3f(V[0].x, V[0].y, V[0].z);
   glVertex3f(V[3].x, V[3].y, V[3].z);
   glVertex3f(V[2].x, V[2].y, V[2].z);
   glVertex3f(V[1].x, V[1].y, V[1].z);
glEnd();
```

Fig 4.5 다각형을 그리는 OpenGL 코드

4.2 조명 설정

생성된 물체를 볼 수 있다는 것은 빛이 존재해야만 가능하며 따라서 가상공간에 빛을 만들고 이를 설정하여야 한다. 어떤 물체의 특정 위치의 점의 색상이라는 것은 그 공간에 있던 조명에서 출발한 빛이 그 물체 표면에 도달하여 물체의 재질과의 작용에 의하여 반사되어 카메라로 들어오는 빛의 색상이다. 그러므로 물체의 색상은 조명, 물체 표면, 관찰자와의 관계에 의해서 결정된다. 먼저 조명에는 크게 방향광원, 점광원, 조명광원 등이 있다. 방향광원은 태양광처럼 일정한 방향으로만 빛이 진행하게 되므로 그 광원의 위치는 의미가 없다. 반면, 점광원은 그 광원이 위치에서 모든 방향으로 동일한 세기를 내 보낸다. 따라서 점광원에서는 조명의 방향이 의미가 없어진다. 조명광원은 특정 위치에서 특정 방향을 향해 쏘는 것이다. 그리고 광원의 폭도 조절 가능하다. 컴퓨터그래픽스에서 사용되는 빛은 red, green, blue의 조합으로 표현된다. 각 성분은 0.0에서 1.0 사이의 값으로 사용된다. 만약 (r, g, b)=(1.0, 1.0, 1.0)라면 그 빛은 흰색이며, (r, g, b)=(1.0, 1.0, 0.0)이면 노란색, (r, g, b)=(0.0, 0.0, 0.0)이면 검정색이다. 물체의 임의의 지점의 색이라는 것은 그 지점으로 들어 온 빛이 반사되어 카메라에 들어오는 빛의 색상이라 하였는데, 먼저 그 지점으로 나아가는 빛의 종류와 반사되는 빛의 종류를 살펴보자. 실세계 공간에서는 무한히 많은 빛이 존재한다. 그러나 컴퓨터그래픽스에서는 그러한 무한히 많은 빛을 모델링하기에는 너무나 많은 시간이 필요하기 때문에 간단하게 처리한다. 물체의 지점으로 들어오는 빛은 두 가지 빛, 간접적으로 오는 빛과 광원으로부터 직접적으로 와서 닿는 빛으로 나눌 수 있다. 간접적으로 오는 빛은 물체들끼리 반사되어 오는 빛, 여러 번 반복해서 반사되는 것 등을 총망라하는 것으로 이들을 모두 계산하기에는 힘들기 때문에 그냥 이 공간에 간접적인 빛이 존재한다고 가정하게 된다. 그래서 조명에서도 이러한 간접적인 빛에 어느 정도는 기여한다고 보고 이러한 광원의 빛의 요소를 ambient light라고 한다. 광원의 ambient light가 모든 물체에 영향을 미치게 되는데 물체의 재질에 따라 반사되는 정도가 달라져서 ambient reflection light가 형성되게 된다. 물체의 지점에 직접적으로 비추어지는 빛은 물체의 표면과 어떤 각도로 이루고 있느냐에 따라 지점에 닿는 빛의 세기 정도가 달라진다고 볼 수 있다. 따라서 광원의 위치와 물체의 그 지점에서의 법선벡터와 상관관계에 의해서 세기 정도가 진다. 이렇게 직접적으로 입사된 빛은 두 가지 요소의 빛으로 반사된다. 하나는 난반사인 diffuse refection light인데 이 반사는 모든 방향으로 향하게 되며, 다른 하나는 거울 반사 놀이에서 나타나는 반사로서 정반사라고 하는 것으로 Specular refection light 인데 입사각과 같은 크기의 반사각을 이루면서 반사되는 것이다. 따라서 물체의 특정 지점의 색상이라는 것은 그 지점에서는 반사되는 ambient light + diffuse light + specular light의 합으로 시뮬레이션된다.

Diffuse Light　　Specular Light　　Ambient Light



공간에 설치된 광원은 전기가 공급이 되어야만 작동을 할 수 있다. OpenGL에서도 이러한 역할을 하는 함수가 있다.

```
glEnable(GL_LIGHTING);
```

전원을 끊는 함수는 glDisable(GL_LIGHTING)이다. 전원이 공급되었다 하더라도 OpenGL에서는 무한 많은 광원을 사용할 수 없다. 사용할 수 있는 광원의 수가 8개로 제한되어 있다. 전원사정이 그렇게 좋지 않는가 보다. 8개의 광원의 이름은 이렇게 정해져 있다.

```
GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, … , GL_LIGHT7
```

자 이제 LIGHT0 광원이 빛이 나게 하기 위해서 스위치를 켜도록 하자.

```
glEnable(GL_LIGHT0);
```

그런데 사용되는 광원이 어디에 있는지, 어떤 색을 비추게 되는지 등에 관한 광원의 특성을 설정하는 함수가 필요하다.

```
ambient_light[] ={1.0, 0.0, 0.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);
```

glLightfv 함수의 첫 번째 인자는 광원의 이름이며, 두 번째 인자는 광원의 특성이며, 세 번째 인자는 광원의 특성으로 사용되는 값을 의미한다. 위 함수를 적용하며, LIGHT0의 광원의 ambient 요소를 빨강색으로 설정한다는 의미이다.