Computer Graphics Term Project
3D Video Game

Final Report

과 목 명 : 컴퓨터그래픽스

1. 프로젝트 목적 및 변경 사항

1) 프로젝트 목적

OpenGL을 이용한 3D비디오 게임 제작

■ 필수 요구 사항

- 3D view and Object
- User interaction
- Texture Mapping
- Lighting and Transparency
- Game techniques: Collision detection, AI, etc.

2) 변경사항

■ 게임의 목적 변경

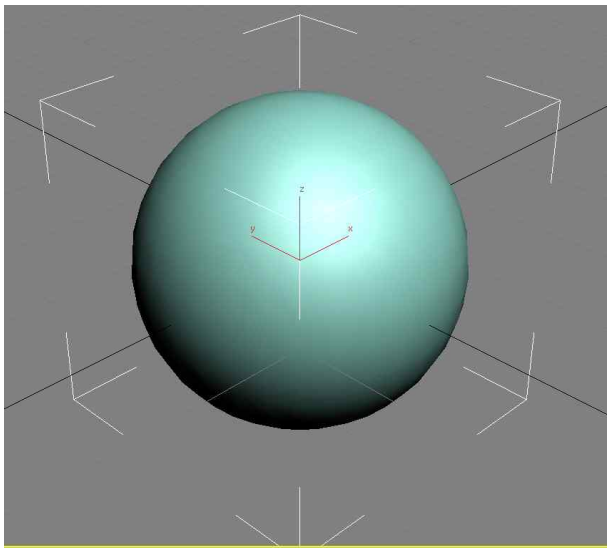
- 이전 중간보고서의 경우 **"날아오는 장애물을 피해 오래 버티기"** 였다.
- 하지만 구현 중 단순히 오래 피하는 것 보다 장애물과 더 많이 충돌하여 점수를 올리는 것이 더 동적이고 게임적 요소가 강하다 생각하여 목적을 수정.
- 최종 게임의 목적은 ***"일정 시간 동안 장애물과 많이 충돌하기"*** 이다.
- 주제를 바꿈으로써 좀 더 동적으로 사용자 interaction을 반영 할 수 있고, 기술적으로도 충돌 판별 기능을 더 특화 시킬 수 있었다.

■ 시나리오

- 사용자는 게임을 시작한다.
- 화면가운데 주인공인 비행기가 놓여있다.
- 비행기를 향하여 수많은 장애물들이 날아 온다.
- 사용자는 마우스와 키보드로 비행기를 움직여 장애물과 충돌한다.
- 점수와 시간은 실시간으로 화면에 표시된다.

■ 공간 한계 범위

- 기존의 제안서에는 3D 큐브 공간이라 하였지만 이를 3D구 공간으로 변경한다.



2. 구현결과

1) 구현환경

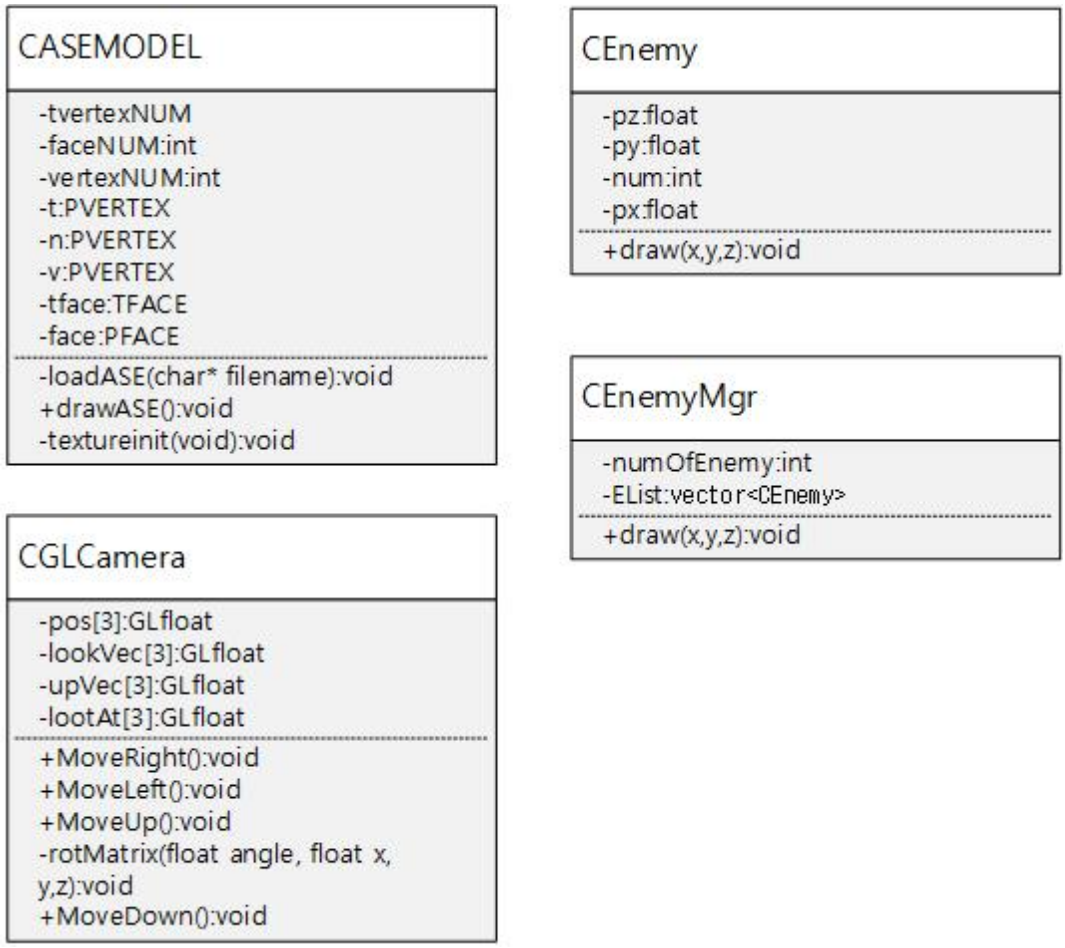
- OS: Microsoft Windows 8.1 Pro 64bit
- 사용 툴: Microsoft Visual Studio 2010, Autodesk 3Ds MAX 9
- 사용 API: win32 API, GLUT, openGL
- Language : C++

2) 프로젝트의 특징

- 자유로운 1인칭(FPS게임 형식) 카메라를 구현 하였다.
- 3D 모델링툴을 이용하여 모델링을 하고 그 결과물인 ASE파일을 파싱하여 Object 를 생성함으로써 좀 더 정교한 모델을 얻을 수 있다.
- 객체지향

3) Structure

■ 주요 구현 Class



① CASEMODEL

- ASE파일을 파싱하여 3D모델을 생성하는 Class이다. 이렇게 생성된 객체는 한 객체당 하나의 모델이 된다.
- 모델을 이루는 버텍스, Normal 벡터, 텍스처 좌표, 제질 값들을 저장 할 수 있는 필드들이 있다.

② CGLCamera

- 1인칭 카메라를 Class화 시킨 것이다. 카메라의 up벡터, left벡터, look벡터 등을 포함 하고 있으며, 상하 좌우로 움직이고, 회전 할 수 있는 메소드들이 정의 되어 있다.

③ CEnemyMgr

- 생성된 장애물들을 관리하는 클래스이다.

- c++ STL의 vector를 이용하여 장애물을 관리한다.

- 총 장애물들의 수가 기록되는 필드가 있다.

- 장애물들을 출력하는 draw메소드가 존재한다.

④ CEnemy

- 장애물을 표현한 Class이다.

- 장애물들의 좌표나타내는 필드인 px, py, pz가 있다.

- 장애물의 번호인 num이 존재한다.

- 장애물을 그리는 draw메소드가 존재한다.

4) 주요 구현 사항

■ ASE파일로부터 3D모델 생성하기 (MODEL ,TEXTURE Mapping)

```
        *TM_ROTAXIS 0.0000      0.0000  0.0000
        *TM_ROTANGLE 0.0000
        *TM_SCALE 1.0000        1.0000  1.0000
        *TM_SCALEAXIS 0.0000    0.0000  0.0000
        *TM_SCALEAXISANG 0.0000
}
*MESH {
        *TIMEVALUE 0
        *MESH_NUMVERTEX 530
        *MESH_NUMFACES 1024
        *MESH_VERTEX_LIST {
                *MESH_VERTEX    0       22.2771 33.8096 12.1157
                *MESH_VERTEX    1       22.1785 33.8096 12.4884
                *MESH_VERTEX    2       22.2928 33.8096 12.6127
                *MESH_VERTEX    3       22.5255 33.8096 12.4884
                *MESH_VERTEX    4       22.7819 33.8096 12.1157
                *MESH_VERTEX    5       21.7293 31.0356 12.1157
                *MESH_VERTEX    6       21.6384 31.0743 12.4884
                *MESH_VERTEX    7       21.7439 31.0294 12.6127
                *MESH_VERTEX    8       21.9585 30.9381 12.4884
                *MESH_VERTEX    9       22.1950 30.8375 12.1157
                *MESH_VERTEX    10      20.2275 28.7917 12.1157
                *MESH_VERTEX    11      20.1575 28.8617 12.4884
                *MESH_VERTEX    12      20.2387 28.7805 12.6127
                *MESH_VERTEX    13      20.4039 28.6153 12.4884
                *MESH_VERTEX    14      20.5859 28.4333 12.1157
                *MESH_VERTEX    15      17.9835 27.2898 12.1157
                *MESH_VERTEX    16      17.9448 27.3808 12.4884
                *MESH_VERTEX    17      17.9897 27.2753 12.6127
                *MESH_VERTEX    18      18.0811 27.0606 12.4884
                *MESH_VERTEX    19      18.1817 26.8241 12.1157
                *MESH_VERTEX    20      15.2095 26.7421 12.1157
                *MESH_VERTEX    21      15.2095 26.8407 12.4884
                *MESH_VERTEX    22      15.2095 26.7263 12.6127
                *MESH_VERTEX    23      15.2095 26.4936 12.4884
                *MESH_VERTEX    24      15.2095 26.2373 12.1157
                *MESH_VERTEX    25      12.2439 27.2898 12.1157
                *MESH_VERTEX    26      12.3934 27.3808 12.4884
                *MESH_VERTEX    27      12.4054 27.2753 12.6127
                *MESH_VERTEX    28      12.3350 27.0606 12.4884
                *MESH_VERTEX    29      12.2374 26.8241 12.1157
                *MESH_VERTEX    30      10.0212 28.7917 12.1157
                *MESH_VERTEX    31      10.1897 28.8617 12.4884
                *MESH_VERTEX    32      10.1591 28.7805 12.6127
                *MESH_VERTEX    33      10.0125 28.6153 12.4884
                *MESH_VERTEX    34      9.8332  28.4333 12.1157
                *MESH_VERTEX    35      8.6259  31.0356 12.1157
```

<ASE파일의 일부>

실제 게임에서는 단순한 모양의 모델보다 좀 더 복잡하고 사실적인 모델을 사용하게 된다. 이 때 이러한 모델들을 모델링 하는 3D모델링 툴이 있다. 예를 들면 3Ds Max , Maya등이 있다. 이러한 모델링 툴로 모델을 모델링함으로써 일일이 버텍스좌표를 찍어 모델링을 하지 않아도 되며 좀 더 복잡한 모델 까지 정확하게 그려 낼 수 있게 된다. 그리고 모델링 결과를 여러 형태의 파일로 export하여 다른 곳에서도 사용 할 수 있게 하고 있다. 프로젝트 수행 시 그 파일 형식 중 아스키형식 파일인 "ASE"파일을 이용하였다.

이 파일에는 오브젝트의 제질 특성 값, 각 면을 이루는 버텍스와, 버텍스의 Normal값, 텍스쳐, 텍스쳐 좌표 등이 일일이 하나하나 매치되어 기록 되어있다. 하지만 그 양이 상당하였다. 흔히 샘플로 쓰이는 주전자를 export하였을 때 버텍스 개수만 500여개가 되었고 이에 매칭되는 Normal값도 있으며 텍스쳐 좌표도 있으니

양이 엄청났다. 일일이 손수 수천라인의 텍스트를 열어 버텍스와 노멀 좌표를 입력하는 것도 한계가 있었다.

그리하여 이 파일을 파싱 할 수 있는 함수를 구현하게 되었다. 더 나아가 이렇게 읽어들인 정보들을 하나의 오브젝트로 보고 객체화 시켜 생성과 이동이 간편하게 하였다. 객체의 생성은 ase파일명으로 생성할 수 있다. 이렇게 생성된 객체를 drawASE()라는 함수를 호출하여 화면에 그릴 수 있다.

```
void CASEMODEL::drawASE()
{


        glPushMatrix();

        glEnable (GL_TEXTURE_2D);

        glBindTexture (GL_TEXTURE_2D, ids[0]);
        glBegin(GL_TRIANGLES);

        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_amb);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_spc);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_dif);


        for(int j=0; j<faceNUM;j++)
        {
            for(int m=0;m<3;m++)
            {

                glNormal3f(face[j].n[m][0],face[j].n[m][1], face[j].n[m][2]);
                glTexCoord2f(t[tface[j].tvertnum[m]].v[0],t[tface[j].tvertnum[m]].v[1]);
                glVertex3f(v[face[j].vertexnum[m]].v[0], v[face[j].vertexnum[m]].v[1],v[face[j].vertexnum[m]].v[2]);
            }

        }
        glEnd();

        glDisable(GL_TEXTURE_2D);

        glPopMatrix();

}
```

위 코드는 drawASE함수이다. for문을 실행하며 각 면에 해당하는 버텍스들의 노멀 값과 텍스처좌표, 버텍스좌표를 찍고 있다.

■ 1인칭 카메라 구현(Viewing)

3D게임에 있어 카메라는 중요하다. 사용자에게 게임을 보여주는 이상의 의미를 가지고 있다. 카메라는 사용자의 입력을 받아들여 그에 해당하는 화면을 보여주는 사용자와 게임을 이어주는 역할을 하고 있다. 아무리 잘 구현된 게임이라도 한 화면만 비춰준다면 사용자는 게임을 즐길 수 없을 것이다.

그리하여 이번 프로젝트에서 좀 더 자유로운 1인칭 카메라를 구현하였다. 비행기를 조종하며 자유 공간을 날아다녀야 하기에 1인칭 시점 카메라가 적합하다 생각한다. 하지만 이러한 자유로운 카메라를 구현하려면 많은 벡터연산과 수학적 지식이 동원되어야 한다. 예를 들어 pitch나 yaw같은 회전 기능의 경우 많은 수학적 이해가 필

요로 하게 된다. 프로젝트 기간이 너무 촉박하여 이를 다 이해하기란 어려웠다. 그리하여 각종 문헌을 참고를 해야 했다. 알맞은 기능을 구현해놓은 프로그램에서 소스를 분석해 클래스화(CGLCamera)시켰다.

또한 카메라는 사용자의 입력에 반응하여 동작해야하기에 Windows API 라이브러리도 많이 참고해야하는 부분이 였다.

카메라는 AWSD의 키에 반응하여 각각 좌, 전진, 후진, 우, 을 하게 된다. 또한 마우스의 움직임으로 회전을 표현하게 된다.

다음으로 카메라 객체를 생성하고 뷰잉을 할 수 있다.

```
cam=CGLCamera();  // 카메라 객체 생성
cam.DrawGLScene(); // 뷰잉
```

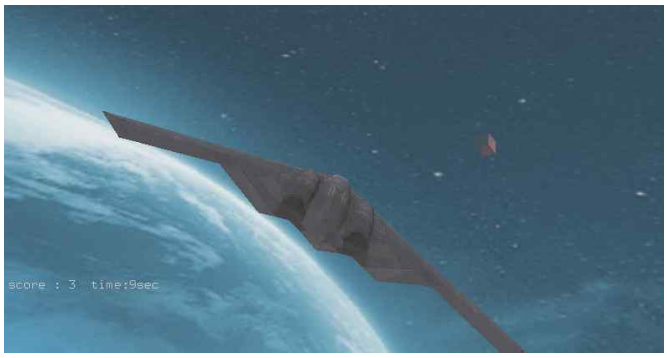
■ User Interaction

User Interaction은 사용자가 실제로 게임을 가능하게 해준다. 때문에 직관적 이여야 하고 단순해야 한다. 이번 프로젝트에서 User Interacton은 앞서 설명한 1인칭 카메라와 연동 되어 작동하게 된다.

① 키보드입력

```
void keycallback(unsigned char key, int x, int y)
{
    printf("%d ",key);


    if(key=='a'){
        plane_rotate_Z=30;
        cam.MoveLeft();
    }
    if(key=='d'){
        plane_rotate_Z=-30;
        cam.MoveRight();
    }
    if(key=='s'){
        cam.MoveDown();
    }
    if(key=='w'){
        cam.MoveUp();

    }

}
```

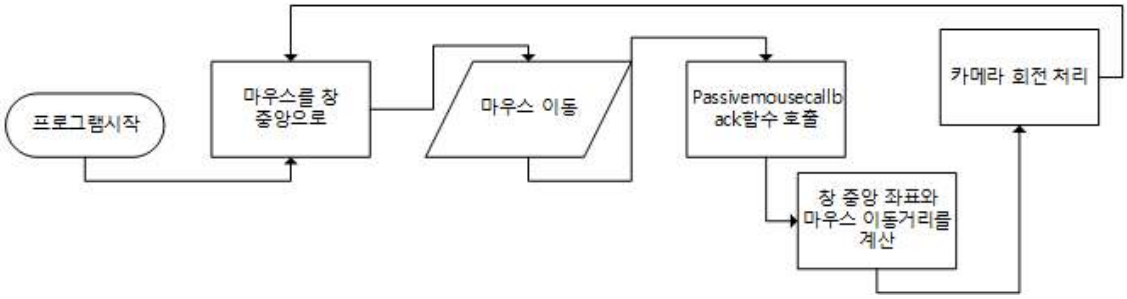
키보드입력의 경우 a,d,s,w의 입력을 가진다 각 키는 카메라의 이동을 조작하며, 게임의 효과를 위해 비행기의 좌우 회전을 컨트롤 한다.



d키를 눌렀을 때 비행기가 기울어진다.

② 마우스 입력

마우스 입력의 경우 키보드 입력보다 더 복잡하였다. 마우스의 이동 거리에 따라 카메라를 회전해야 하므로 마우스를 게임 창 중앙으로 고정 시켜야 했다.



그 처리 순서도는 위와 같다. 또한 아래 코드는 마우스 처리를 위한 glut에서 제공하는 메소드이다.

```
glutSetCursor(GLUT_CURSOR_NONE); //마우스 숨김
glutWarpPointer(WINSIZEX/2,WINSIZEY/2); //마우스 화면 중앙으로 이동
```

■ 장애물 생성 및 이동

게임의 중요한 요소인 적 혹은 장애물. 그것을 구현하기 위해 이번 프로젝트에서는 클래스 설계로 좀 더 쉽게 생성해 볼 수 있었다. 장애물 하나를 Enemy Class 그리고 장애물은 하나만 있는게 아니라 다수 이므로 이들을 관리 할 수 있는 EnemyMgr라는 Class를 정의 하였다.

```
emgr=CEnemyMgr(500,&Score); //500개의 장애물 생성
```

장애물은 다음과 같이 EnemyMgr이라는 객체를 생성함으로써 동시에 생성이 된다.

```
CEnemyMgr::CEnemyMgr(int n,int* Score)
{
    this->Score=Score;
    enemys=EList();
    this->collisionScore=0;
    srand((unsigned) time(NULL));
    for(int i=0;i<n; i++)
    {
    //  CEnemy a;
         int a = -(-300 + (int)(600 * rand() / (RAND_MAX + 1.0)));
         int b = -(-300 + (int)(600 * rand() / (RAND_MAX + 1.0)));//rand()%300;
         int c =  -(-300 + (int)(600 * rand() / (RAND_MAX + 1.0)));
        // printf("%d ",a);
        enemys.push_back(CEnemy(a,b,c,i,Score));
    }

}
```

위 소스는 EnemyMgr 객체의 생성자이다. 랜덤함수를 이용하여 Enemy객체의 좌표를 임의로 지정해 생성하고 있다.

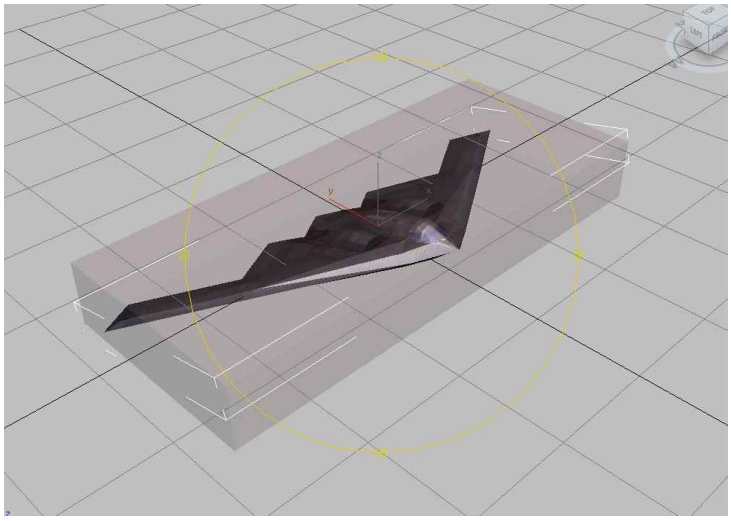
이렇게 생성된 장애물들은 display 콜백 함수에서 CEnemyMgr객체의 draw메소드를 호출함으로써 화면에 출력 될 수 있다.

```
void CEnemyMgr::draw(float vX, float vY, float vZ)
{


    for(int i=0;i< enemys.size(); i++)
    {

        enemys.at(i).draw(vX, vY, vZ); //목표지점 포지션..

    }


}
```

이 때 C++의 STL의 vector 구조체를 이용함으로써 index로 쉽게 접근이 가능하도록 처리하였다. 또 한 enemy의 draw함수를 호출할 때 좌표를 하나 주는데 이는 비행기의 좌표로 이를 통해 앞서 설명 할 충돌 검사를 할 수 있도록 한다.

■ 충돌 검사

충돌검사의 경우 처음 접해보는 방법이라 어려움이 많았다. 하지만 조사결과 바운드 박스라는 기법을 사용하여 구현하는 방법이 많아서 적용해 보았다.



실제로 위 그림처럼 박스를 씌우진 않는다. 박스가 씌워진 것처럼 생각하여 최대좌표와 최소 좌표를 각각 xyz축으로 구한 후 장애물이 그 범위 안에 들어가면 충돌로 판별하는 방법을 사용하였다.

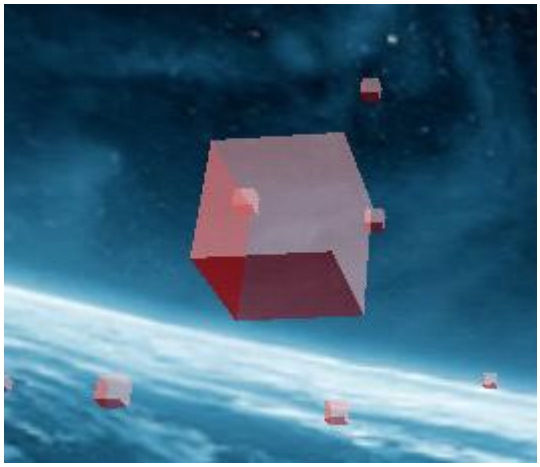
위에서 설명할 때 Enemy의 draw함수를 호출할 때 좌표를 넣어준다고 설명하였는데 이 좌표가 바로 비행기의 현재 좌표이다. Enemy객체는 이 좌표를 가지고 최대좌표와 최소 좌표를 계산해 낸다. 다음 자기 자신의 위치가 그 안에 있으면 충돌로 판단 하여 반응을 하게 되는 것이다.

```
if(
    tx-45<px&&px<tx+45&&
    ty-15<py&&py<ty+15&&
    tz-20<pz&&pz<tz+20 &&isCollision==0
    )
    {
        printf("%d번 장애물 충돌!!!!!!!!₩n",this->num);
        //if(isCollision=!1)
            isCollision++;
            isReset=0;
            (*Score)++;
    }
```

위 소스는 충돌 판단을 하는 조건을 나타낸 코드이다.

■ 장애물 Blend효과

장애물에게 투명도를 주어 좀 재미있는 효과를 연출해 보았다.



```
GLfloat mat_amb[4]={0.5, 0.0,0.0, 0.5}; // 주변광에 대한 반사값
GLfloat mat_dif[4]={1,0.0, 0.0, 0.5};   // 분산광에 대한 반사값
GLfloat mat_spc[4]={0.7, 0.7, 0.7, 1.0};    // 반사광에 대한 반사값
GLfloat mat_emi[4]={0.0,0.0, 0.0, 0.0}; // 발광값
GLfloat mat_shi[1]={0.0};   // 광택
```

재질 속성 값에 알파값으로 투명도를 조절하였다. 또한 display 콜백 함수에서 불투명한물체를 먼저 그린 후 투명한 물체를 그리고 있다.

```
drawPlanet( );

glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);


glPushMatrix( );
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_amb);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_spc);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,  mat_dif);

    emgr.draw(cam.GetLookAtVec( )[0],cam.GetLookAtVec( )[1]-15,cam.GetLookAtVec( )[2]-30);


glPopMatrix( );
```

■ **타이머 구현**

게임의 끝은 있어야하고 점수를 내려면 더욱이 제한 시간이 필요하다. 때문에 본 프로젝트에서는 windows API의 setTimer함수를 이용해 타이머를 제한 시간을 구현하였다.

```
VOID CALLBACK TimerProc(HWND hWnd, UINT nMsg, UINT nIDEvent, DWORD dwTime) {


    sec+=1000;
    wchar_t msg[100];
    wsprintf(msg,TEXT("당신의 점수는 %d 점입니다."),Score);
    if(sec==30*1000)
    {
        KillTimer(NULL, TimerId);
        MessageBox(NULL,msg,TEXT("게임오버"),NULL);
        exit(1);
    }

}
```

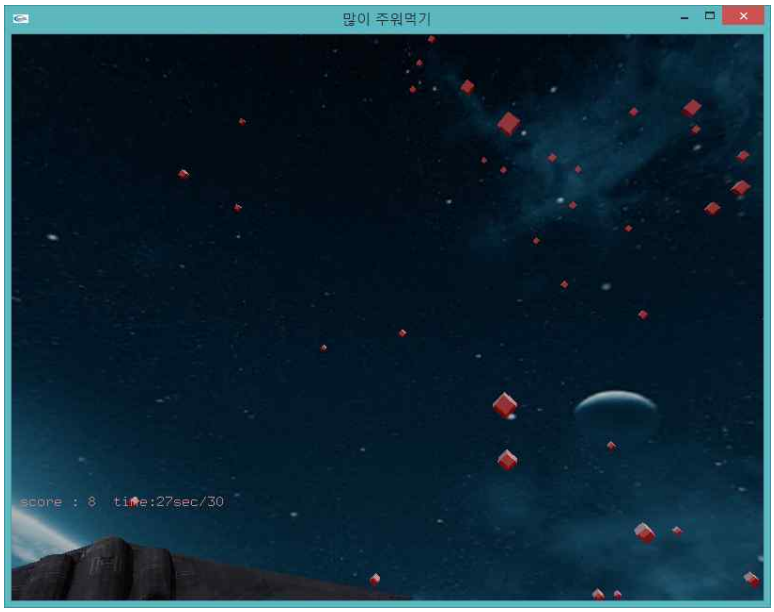
타이머는 1초에 한번씩 위 콜백함수를 호출한다. 그리고 30초가 지나면 타이머를 중기하고 게임을 중단시키며, 점수를 출력한다.

3. 테스트

■ 게임 실행

게임의 첫 화면이다. 게임시작 시 비행기는 중앙에 있으며 장애물들은 생성이 되어 Z축방향으로 날아오게 된다. 좌측하단에 시간과 점수가 표시된다.

■ 자유로운 시점이동



마우스 이동을 통하여 자유로운 시점이동이 가능하다.

■ 장애물 충돌

장애물과 충돌시 장애물은 큐브형태에서 구 형태로 변화한다.

■ **게임오버**

시간 초과 시 점수를 알려주는 메시지 박스를 출력한다.

4. 후기

그래픽스과목은 대학교에 들어오기 전부터 관심을 갖고 싶었던 과목이 였다. 그래서 3d모델링 관련 툴도 만져보던적이 있었다. 그때 어렴풋이 알던 개념들이 이번 그래픽스 과목을 통하여 이해가 확실해 졌다. 또한 전에 해보았던 것들 때문인지 그래픽스 과목도 어렵지만 나름 재미가 있던 과목이 였다. 프로젝트를 하면서 이번 학기 배웠던 개념들을 총망라하여 구현한 것 같다. 라이팅, 텍스처, 알파블렌딩, 카메라 등의 개념이 모두 사용된 것 같다. 또한 게임이라는 프로그래밍을 처음 경험해 보았다. 게임프로그래밍은 이전의 네트워크나 시스템 프로그래밍과는 좀 다른 분야 같다는 생각이 든다. 처리한 것을 반드시 화면에 출력해야 하며, 눈에 보이기 위해서 엄청난 수학적 처리가 들어가는 것도 알게 되었다. 비록 타 과목 기말고사와 텀프로젝트와 겹쳐 여유를 가지고 개발해 볼 수가 없어 아쉬웠지만 또 다른 분야를 경험하여 많은 것을 깨우친 프로젝트였던 것 같다.

※ 참고문헌

[1] 1인칭 카메라, http://kgw1988.blog.me/90174514597

[2] 화면에 TEXT 출력하기 , http://rbellek.wordpress.com/2011/03/05/51/