컴퓨터 그래픽스 최종보고서 – 20163369 이효은

주제 – 스카이로드(Skyloads) / 우주선 게임

구현 내용

- 전체 구현내용

1) Skybox를 사용하여 텍스쳐로 전체적인 배경을 설정, 라이브러리 설정 문제는 #pragma comment (lib, "legacy\_stdio\_definitions.lib")선언, bmp 파일을 인식하지 못하는 문제는 유니코드->멀티바이트 문자 집합으로 설정 변경 후 사용으로 해결

2) glutSpecialFunc 을 사용하여 위쪽 방향키 사용 시 속도 증가, 아래쪽 방향키 사용 시 속도 감소, 왼쪽&오른쪽 방향키 사용 시 그 방향으로 우주선을 이동시킴. 속도의 경우 일정 속도 이상으로 증가되지 않도록 처리되었음

3) glutKeyboardFunc 을 사용하여 스페이스 바 사용 시 우주선을 점프시킴, 임시로 glutTimerFunc 을 사용해 일정 시간동안 우주선의 y좌표를 증가시키는 방법으로 구현

4) s버튼 혹은 메뉴의 "BGM" -> "배경음악 재생/정지" 를 누를 시 배경음을 재생하거나 정지

5) 메뉴의 "BGM" -> "음량 증가", "음량 감소" 를 누를 시 배경음의 음량 조절 가능, FMOD의 기능을 사용하여 구현하였음 (pChannel[0]이 배경음의 주소)

6) r버튼 혹은 메뉴의 "Reset" 을 누를 시 게임 재시작

7) esc버튼 혹은 메뉴의 "Exit" 를 누를 시 게임 종료

8) 맵의 경우 ground라는 함수를 선언, 사용하여 너비와 길이만 정해주면 화면에 출력되도록 작성하였음, 단 맵의 위치는 glTranslatef를 사용하여 직접 이동시켜 주었으며 이로 인해 맵 전체의 코드 길이가 길어져 별개의 함수로 작성되었음.

9) 우주선은 glutSolidTeapot, gluCylinder, glutSolidSphere 를 조합하여 구현되었음.

- 계획서 달성내용 (달성한 것은 O, 미완성은 ?, 아닌 것은 X로 옆에 표기)

1) 우주를 모티브로 한 배경에 넓은 직사각형들로 구성된 길들이 펼쳐지고 우주선이 등장함. **(O)**

-> 텍스처 기능을 활용한 Skybox를 사용하여 배경 처리 완료, 길들과 우주선도 구현 완료

2) 위&아래 방향키로 우주선이 앞으로 나아가는 속도 조절**(O)**

-> 타이머 콜백과 bool 변수를 사용하여 구현

3) 오른쪽&왼쪽 방향키로 양 옆으로 움직이게 함**(O)**

-> 우주선의 좌표값을 전역 변수인 배열 처리하여 방향키 입력마다 값이 직접 바뀌게 해서 움직임 구현.

4) 스페이스키로 우주선을 점프시킴. **(?)**

-> 미완성, 점프는 가능하나 매끄러운 착지가 불가능하고 타이머 콜백의 한계로 인해 키 입력이 무시되는 문제 발생, 스페이스바 연타로 공중에 띄울 수 있어 이로 게임 클리어 가능

5) 길 바깥의 허공으로 떨어지면 게임 오버 **(X)**

-> 구현 실패, 길이 맞는지 아닌지 판정할 방법을 찾지 못했음

6) 가던 중 장애물에 가로막힐 시 멈춤(속도가 0이 됨) **(X)**

-> 구현 실패, 충돌 판정 방법은 찾았으나 장애물의 정확한 좌표값을 구할 방법을 찾지 못했음

7) 지정된 장소에 도착 시 게임 끝 **(?)**

-> 미완성, 일단 지정된 좌표에 도착 시 콘솔에 Game End 출력, 비행기는 멈추고 화면만 움직이게 해서 마지막을 표현했으나 주변 허공에 대한 처리가 이루어지지 않음.

+) 메뉴나 키 입력을 사용하여 리셋 기능, 종료 기능 삽입 -> 메뉴와 키 입력 모두 구현 완료 **(O)**

화면 윗부분에 현재 속도, 현재까지 소요된 시간과 도착까지 남은 거리를 바 형태로 표시 -> 구현 실패 **(X)**

콘솔창에서는 게임 방법을 출력 -> 구현 완료 **(O)**

- 계획서 외의 달성내용

FMOD 함수를 사용하여 배경음을 삽입하였음.

- 계획서와 달라진 점

1) 길을 입체도형이 아닌 평면을 회전시켜 구현하였음.

2) 원 계획서에서는 길의 높이가 달라지기도 하지만, 길의 판정 문제 때문에 실제 구현에는 높이가 모두 같도록 설정되었음.

3) 현재 속도와 소요된 시간, 도착까지 남은 거리를 뷰포트 분할을 사용하여 위에서 구현하려 했으나 오류로 인해 미구현되었음.

4) 조명이 불완전하게 설정되어 제대로 드러나지 않았고 이로 인해 그림자 구현이 이루어지지 못함.

5) 길의 유무 판정을 구현하지 못해 우주선이 길 외의 곳에 있을 때 허공으로 떨어지는 처리도 이루어지지 않음.

6) 점프가 부드러운 포물선을 그리지 못했고 좌표값 이동만 구현되었음

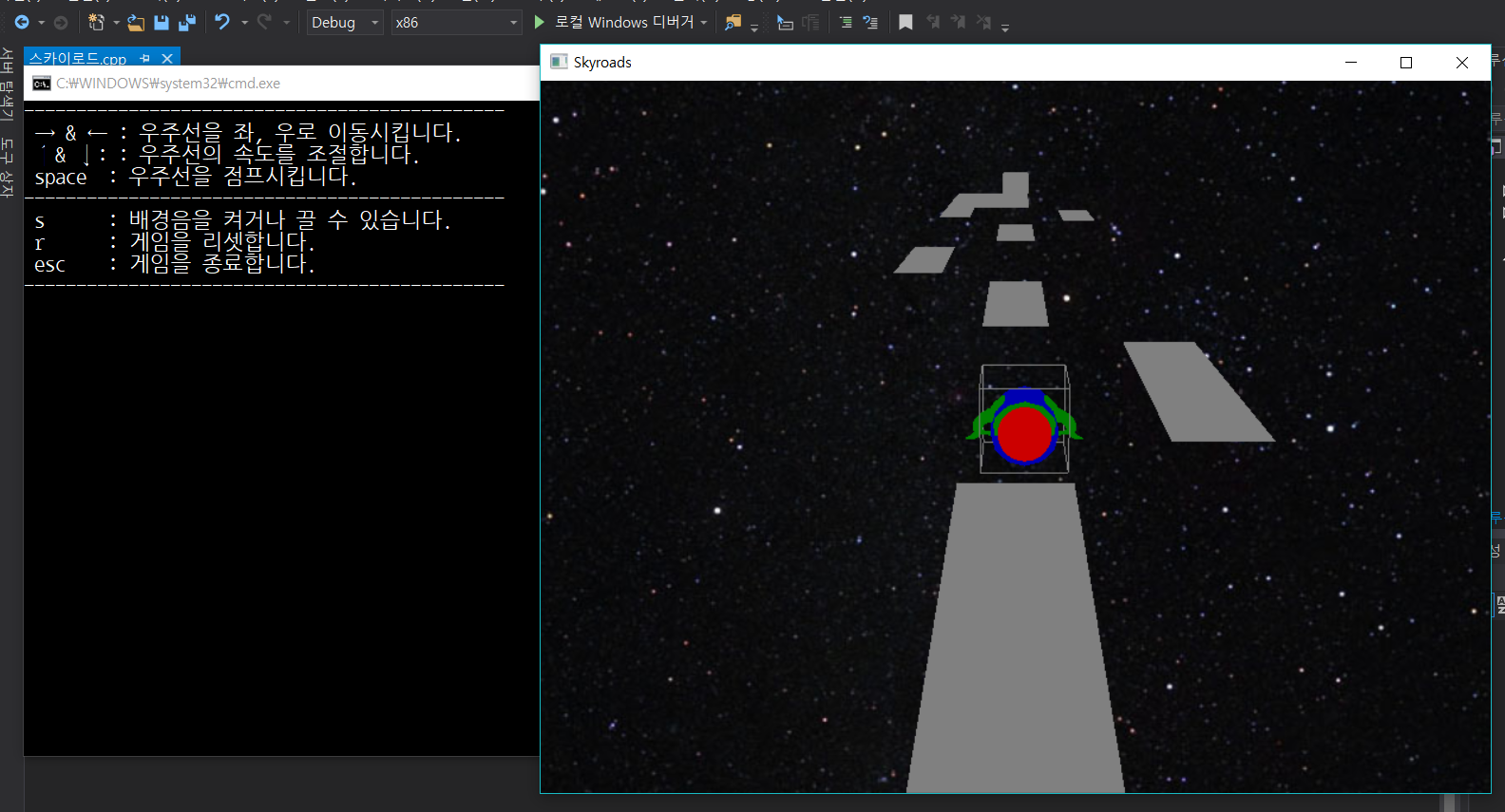
소감

여러모로 힘들었던 과제였다. 이상은 컸지만 현실은 시궁창이라는 것을 절실히 느꼈던 그런 과제이기도 했다.

코드만 작성 해 두고 집어넣지 않은 기능들이 많아 안타깝지만 제출 이후에도 차차 수정해 나갈 예정이다.

추락 판정을 마지막까지도 구현하지 못한 것(구현하기 위해선 다른 알고리즘을 사용하거나 맵 전체의 좌표값과 우주선의 현재 좌표값과의 충돌 연산을 구현해야 했다. 후자는 시작 위치와 극 좌표계 설정 등으로 인해 복합적으로 좌표가 꼬여버렸고 맵이 커지면서 기간 내에 처리하기란 불가능해졌다.)과 점프가 부드럽게 되지 않았던 것, 조명의 기본 코드는 작성했으나 실제로 사용되진 않았고 임시 색조합을 마지막까지 사용한 것, 그리고 맵을 직접 설게 해 놓고도 실제 구현은 일부만 됐던 것이 제일 아쉬웠던 것 같다.

구현했던 것 중에서 가장 골치아팠던 것은 역시 배경의 텍스쳐 부분이었는데, 조명처럼 활성화-비활성화를 해주면서 작성해야 했던 것도 처음엔 몰랐기 때문에 헤맸으며, 어차피 큰 시점변환이 없는 게임이라 보이는 부분만 텍스쳐를 처리하려다 우주선의 시점을 이동할 때 마다 배경도 같이 이동한다거나 배경의 기본 위치나 크기가 제대로 정의되지 않는 문제가 있어 찾아보던 중 Skybox를 이용하여 거대한 박스에 게임의 요소들을 집어넣는 형식으로 구현하게 되었다. 그랬더니 앞의 문제도 해결되었고 맵을 확장했을 시 배경이 사라지는 문제도 사라졌으나 미리 설정해둔 시점 덕분에 천장의 텍스쳐 배경이 시야를 가려버리는 문제가 발생하여 결국 천장을 없애는 방법을 사용해 해결했다.

****실행 결과

전체 소스 코드

/\*

날짜 : 2017/05/17 ~ 06/12

작성자 : 이효은

파일명 : 스카이로드.cpp

주제 : 우주선 게임

\*/

#include <glut.h>

#include <glu.h>

#include <gl.h>

#include <iostream>

#include <Windows.h>

#include <glaux.h>

#include <glext.h>

#include <cmath>

#include <mmsystem.h>

#include "fmod.hpp"

#pragma comment (lib, "fmodex\_vc.lib")

#pragma comment (lib, "legacy\_stdio\_definitions.lib")

using namespace std;

using namespace FMOD;

System\* pSystem; //배경음 설정용 전역변수

Sound\* pSound[2];

Channel\* pChannel[2];

bool IsBGPlaying = false;

float fVolume = 0.6;

int MenuChk;

GLint width = 800, height = 600;

GLfloat look1 = 0.0; //우주선과 함께 시선이 이동하도록 해주는 전역변수

unsigned int speed = 5; //우주선의 속도를 조절하는 전역변수

float g\_lightPosition[] = { 0.0f, 4.0f, 16.0f }; //시작 시 우주선의 좌표 X, Y, Z

bool jump = false, over = false, go = false;

//각각 점프여부, 게임오버 여부, 비행기가 앞으로 가는 움직임의 여부이다.

GLfloat g\_light[] = { 0.0, 0.0, 1.5, 1.0 };

GLfloat light0\_ambient[] = { 0.5, 0.4, 0.3, 1.0 };

GLfloat light0\_diffuse[] = { 0.8, 0.7, 0.6, 1.0 };

GLfloat light0\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

GLuint tex[6]; //설정할 텍스처 이미지의 개수

AUX\_RGBImageRec \*LoadBMP(char \*Filename) { // Bitmap 이미지를 호출한다.

FILE \*File = NULL;

if (!Filename) return NULL;

File = fopen(Filename, "r");

if (File) {

fclose(File);

return auxDIBImageLoad(Filename);

}

return NULL;

}

void LoadGLTextures() { // Bitmap 이미지 6개를 호출하여 Texture 이미지로 변환한다.

AUX\_RGBImageRec \*texRec[6];

memset(texRec, 0, sizeof(void \*) \* 6);

if ((texRec[0] = LoadBMP("cosmo.bmp")) &&

(texRec[1] = LoadBMP("cosmo.bmp")) &&

(texRec[2] = LoadBMP("cosmo.bmp")) &&

(texRec[3] = LoadBMP("cosmo.bmp")) &&

(texRec[4] = LoadBMP("cosmo.bmp")) &&

(texRec[5] = LoadBMP("cosmo.bmp"))) {

for (int i = 0; i<6; i++) {

glGenTextures(1, &tex[i]);

glBindTexture(GL\_TEXTURE\_2D, tex[i]);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, texRec[i]->sizeX, texRec[i]->sizeY, 0,

GL\_RGB, GL\_UNSIGNED\_BYTE, texRec[i]->data);

}

}

for (int i = 0; i<6; i++) {

if (texRec[i]) {

if (texRec[i]->data) free(texRec[i]->data);

free(texRec[i]);

}

}

glEnable(GL\_TEXTURE\_2D);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

}

void MyInit(void) { //전체조명과 지역조명 0 설정

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light0\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light0\_specular);

}

void PolarView(GLfloat radius, GLfloat elevation, GLfloat azimuth, GLfloat twist) {

glTranslatef(0.0, 0.0, -radius); //극 좌표계 설정

glRotatef(-elevation, 1.0, 0.0, 0.0);

glRotatef(azimuth, 0.0, 0.0, 1.0);

glRotatef(-twist, 0.0, 0.0, 1.0);

}

void board() { //우주선 출력

glFrontFace(GL\_CW);

GLUquadricObj \*obj;

obj = gluNewQuadric();

gluQuadricDrawStyle(obj, GLU\_FILL);

glPushMatrix();

glDisable(GL\_LIGHTING);

glColor3f(0.0, 0.5, 0.0);

glRotatef(180.0, 0.0, 0.0, 0.0);

glutSolidTeapot(0.4);

glEnable(GL\_LIGHTING);

glDisable(GL\_LIGHTING);

glRotatef(180.0, 0.0, 180.0, 0.0);

glutSolidTeapot(0.4);

glEnable(GL\_LIGHTING);

glDisable(GL\_LIGHTING);

glColor3f(0.0, 0.0, 0.7);

glTranslatef(0.0, 0.0, -0.2);

gluCylinder(obj, 0.38, 0.38, 0.6, 12, 10);

glEnable(GL\_LIGHTING);

glDisable(GL\_LIGHTING);

glColor3f(0.5f, 0.5f, 0.5f);

glutWireCube(1.0);

glEnable(GL\_LIGHTING);

glDisable(GL\_LIGHTING);

glTranslatef(0.0, 0.0, -0.7);

gluCylinder(obj, 0.0, 0.38, 0.7, 12, 10);

glEnable(GL\_LIGHTING);

glDisable(GL\_LIGHTING);

glColor3f(0.8, 0.0, 0.0);

glTranslatef(0.0, 0.0, 1.3);

glutSolidSphere(0.3, 60, 60);

glEnable(GL\_LIGHTING);

glColor3f(1.0, 1.0, 1.0);

glPopMatrix();

}

void ground(float w, float h) { //길들의 기본이 되는 함수 (w = 길의 폭, h = 길의 길이)

glColor3f(0.5f, 0.5f, 0.5f);

glBegin(GL\_QUADS);

glVertex3f(-w, w, 0.0);

glVertex3f(w, w, 0.0);

glVertex3f(w, -h, 0.0);

glVertex3f(-w, -h, 0.0);

glEnd();

glColor3f(1.0, 1.0, 1.0);

}

void map() { //ground함수를 이용하여 게임 맵을 설정함

glTranslatef(0.0f, 3.5f, 16.0f);

glRotatef(90.0f, 1.0f, 0.0f, 0.0f);

ground(1.0, 7.0); //1

glTranslatef(4.0f, -10.0f, 0.0f);

ground(1.0, 7.0); //2

glTranslatef(-4.0f, -10.0f, 0.0f);

ground(1.0, 7.0); //3

glTranslatef(-4.0f, -10.0f, 0.0f);

ground(1.0, 7.0); //4

glTranslatef(4.0f, -10.0f, 0.0f);

ground(1.0, 7.0); //5

glTranslatef(4.0f, -10.0f, 0.0f);

ground(1.0, 6.0); //6

glTranslatef(-8.0f, -2.0f, 0.0f);

ground(1.0, 20.0); //7 - 1

glTranslatef(3.0f, -8.0f, 0.0f);

ground(2.0, 12.0); //7 - 2

glColor3f(0.5f, 0.5f, 0.5f);

glTranslatef(1.0f, -6.0f, -1.0f);

glutSolidCube(2.0); //장애물

glTranslatef(0.0f, -9.0f, 1.0f);

ground(1.0, 12.0); //8

}

void MyDisplay() {

GLfloat LightPosition[] = { 0.0, 0.0, 1.5, 1.0 };

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glDisable(GL\_LIGHTING);

float r = 999.9f;

glPushMatrix();

// Skybox의 앞면을 Rendering 한다.

glBindTexture(GL\_TEXTURE\_2D, tex[0]);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0); glVertex3f(-r, -r / 2, -r);

glTexCoord2f(1, 0); glVertex3f(r, -r / 2, -r);

glTexCoord2f(1, 1); glVertex3f(r, r, -r);

glTexCoord2f(0, 1); glVertex3f(-r, r, -r);

glEnd();

// Skybox의 뒷면을 Rendering 한다.

glBindTexture(GL\_TEXTURE\_2D, tex[1]);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0); glVertex3f(r, -r / 2, r);

glTexCoord2f(1, 0); glVertex3f(-r, -r / 2, r);

glTexCoord2f(1, 1); glVertex3f(-r, r, r);

glTexCoord2f(0, 1); glVertex3f(r, r, r);

glEnd();

// Skybox의 오른쪽 면을 Rendering 한다.

glBindTexture(GL\_TEXTURE\_2D, tex[2]);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0); glVertex3f(r, -r / 2, -r);

glTexCoord2f(1, 0); glVertex3f(r, -r / 2, r);

glTexCoord2f(1, 1); glVertex3f(r, r, r);

glTexCoord2f(0, 1); glVertex3f(r, r, -r);

glEnd();

// Skybox의 왼쪽 면을 Rendering 한다.

glBindTexture(GL\_TEXTURE\_2D, tex[3]);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0); glVertex3f(-r, -r / 2, r);

glTexCoord2f(1, 0); glVertex3f(-r, -r / 2, -r);

glTexCoord2f(1, 1); glVertex3f(-r, r, -r);

glTexCoord2f(0, 1); glVertex3f(-r, r, r);

glEnd();

// Skybox의 윗면을 Rendering 한다.

glBindTexture(GL\_TEXTURE\_2D, tex[4]);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0); glVertex3f(-r, r, -r);

glTexCoord2f(1, 0); glVertex3f(r, r, -r);

glTexCoord2f(1, 1); glVertex3f(r, r, r);

glTexCoord2f(0, 1); glVertex3f(-r, r, r);

glEnd();

// Skybox의 아랫면을 Rendering 한다.

glBindTexture(GL\_TEXTURE\_2D, tex[5]);

glBegin(GL\_QUADS);

glTexCoord2f(0, 0); glVertex3f(-r, -r / 2, r);

glTexCoord2f(1, 0); glVertex3f(r, -r / 2, r);

glTexCoord2f(1, 1); glVertex3f(r, -r / 2, -r);

glTexCoord2f(0, 1); glVertex3f(-r, -r / 2, -r);

glEnd();

glPopMatrix();

glEnable(GL\_LIGHTING);

glDisable(GL\_TEXTURE\_2D);

PolarView(27.0, -20.0, 0.0, 0.0); //극 좌표계 설정

glTranslatef(0.0, 0.0, look1); //시점이 우주선과 함께 움직이도록 설정

glFrustum(-1.0, 1.0, -1.0, 1.0, 0.0, 5.0); //원근 투영

glPushMatrix(); //지역조명 설정

glTranslatef(0.0, 15.0, look1);

glLightfv(GL\_LIGHT0, GL\_POSITION, g\_light);

glDisable(GL\_LIGHTING);

glColor3f(0.9, 0.9, 0.9);

glutWireSphere(0.06, 10, 10);

glEnable(GL\_LIGHTING);

glPopMatrix();

glPushMatrix();

glTranslatef(g\_lightPosition[0], g\_lightPosition[1], g\_lightPosition[2]);

//우주선의 위치 설정

board();

glPopMatrix();

glDisable(GL\_LIGHTING);

map(); //맵 설정

glEnable(GL\_LIGHTING);

glEnable(GL\_TEXTURE\_2D);

glutSwapBuffers();

}

void MyReshape(int NewWidth, int NewHeight) {

glViewport(0, 0, NewWidth, NewHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (GLfloat)NewWidth / (GLfloat)NewHeight, 1.0, 1000.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void reset() { //메뉴에서 reset항목을 선택하거나 r 버튼을 누를 시 실행되는 함수

cout << "게임을 새로 시작합니다." << endl;

g\_lightPosition[0] = 0.0f;

g\_lightPosition[1] = 4.0f;

g\_lightPosition[2] = 16.0f;

look1 = 0.0f;

over = false;

go = false;

speed = 5;

}

void jumpTimer(int Value) { //우주선을 점프시키는 함수

if (jump) {

g\_lightPosition[1] = 6.0f;

jump = false;

}

else { //jump가 true가 아니면 점프를 하지 않는다.

g\_lightPosition[1] = 4.0f;

}

glutTimerFunc(2000, jumpTimer, 1);

glutPostRedisplay();

}

void MyKeyboard(unsigned char KeyPressed, int X, int Y) { //키보드 콜백

switch (KeyPressed) {

case 'r':

reset(); //리셋

break;

case 's':

if (IsBGPlaying) { //현재 음악이 실행되고 있을 경우 음악을 정지한다.

cout << "배경음악을 정지합니다." << endl;

pChannel[0]->stop();

IsBGPlaying = false;

}

else { //현재 음악이 멈춰 있을 경우 음악을 재생한다.

cout << "배경음악을 재생합니다." << endl;

pSystem->playSound(FMOD\_CHANNEL\_REUSE, pSound[0], false, &pChannel[0]);

pChannel[0]->setVolume(fVolume);

IsBGPlaying = true;

}

break;

case 32: //스페이스바(return 32) 입력

jump = true; //위의 jumpTimer함수에서 점프를 할 수 있도록 하는 트리거

break;

case 27:

printf("프로그램을 종료합니다.\n");

exit(0);

//Esc키(return 27) 입력시 콘솔창에 "프로그램을 종료합니다."를 출력한 뒤 프로그램 종료

break;

default:

break; //일반적인 키의 경우 눌러도 반응 x

}

glutPostRedisplay();

}

void MySpecial(int key, int x, int y) {

if (key == GLUT\_KEY\_UP) {

go = true; //우주선을 앞으로 움직이게 한다는 트리거 실행

if (speed >= 140) { //스피드의 값이 140이 넘을 경우 더이상 증가되지 않음

cout << "더 이상 증가시킬 수 없습니다." << endl;

speed = 140;

}

else {

speed += 5; //스피드 증가

}

}

if (key == GLUT\_KEY\_DOWN) {

if (speed <= 5) {

//스피드의 값이 5보다 낮을 경우 더이상 감소되지 않고 우주선을 멈추는 트리거 실행

cout << "더 이상 감소시킬 수 없습니다." << endl;

speed = 5;

go = false;

}

else {

speed -= 5; //스피드 감소

}

}

if (key == GLUT\_KEY\_LEFT) {

g\_lightPosition[0] -= 0.1f; //우주선을 왼쪽으로 이동시킴

}

else if (key == GLUT\_KEY\_RIGHT) {

g\_lightPosition[0] += 0.1f; //우주선을 오른쪽으로 이동시킴

}

glutPostRedisplay();

}

void MyTimer(int Value) {

if (go) { //go 가 참일 경우에만 실행

if (g\_lightPosition[2] > -74.7) {

g\_lightPosition[2] -= 0.1f; //우주선을 Z축 방향으로 이동시킴

look1 += 0.1; //시점도 우주선과 함께 이동

}

else {

g\_lightPosition[2] = -74.7; //맵의 끝까지 도달하면 그자리에서 멈춤

cout << ">>> Game End <<<" << endl;

over = true;

}

}

glutTimerFunc(150 - speed, MyTimer, 1);

glutPostRedisplay();

}

void MenuProc(int entryID) {

MenuChk = entryID;

//glutAddMenuEntry("Reset", 1);

if (entryID == 1) {

reset(); //리셋

}

//glutAddMenuEntry("배경음악 재생/정지", 21);

if (entryID == 21) {

if (IsBGPlaying) {

cout << "배경음악을 정지합니다." << endl;

pChannel[0]->stop();

IsBGPlaying = false;

}

else {

cout << "배경음악을 재생합니다." << endl;

pSystem->playSound(FMOD\_CHANNEL\_REUSE, pSound[0], false, &pChannel[0]);

pChannel[0]->setVolume(fVolume);

IsBGPlaying = true;

}

}

//glutAddMenuEntry("음량 증가", 22);

if (entryID == 22) {

pChannel[0]->getVolume(&fVolume); //음량 값을 가져옴

pChannel[0]->setVolume(fVolume + 0.2f); //음량 값을 증가시킴

cout << "음량을 증가시킵니다." << endl;

}

//glutAddMenuEntry("음량 감소", 23);

if (entryID == 23) {

if (fVolume < 0.2) {

cout << "더이상 감소시킬 수 없습니다." << endl;

}

else {

pChannel[0]->getVolume(&fVolume); //음량 값을 가져옴

pChannel[0]->setVolume(fVolume - 0.2f); //음량 값을 감소시킴

cout << "음량을 감소시킵니다."<< endl;

}

}

//glutAddMenuEntry("Exit", 4);

if (entryID == 4) {

cout << "프로그램을 종료합니다." << endl;

exit(0);

//메뉴에서 Exit 선택시 콘솔창에 "프로그램을 종료합니다."를 출력한 뒤 프로그램 종료

}

glutPostRedisplay();

}

void MenuFunc() {

GLint subMenu2 = glutCreateMenu(MenuProc); //BGM의 하위메뉴

glutAddMenuEntry("배경음악 재생/정지", 21);

glutAddMenuEntry("음량 증가", 22);

glutAddMenuEntry("음량 감소", 23);

GLint mainMenu = glutCreateMenu(MenuProc); //메인 메뉴

glutAddMenuEntry("Reset", 1);

glutAddSubMenu("BGM", subMenu2);

glutAddMenuEntry("Exit", 4);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void SoundIn() {

System\_Create(&pSystem); //사운드 시스템을 생성

pSystem->init(2, FMOD\_INIT\_NORMAL, NULL); //시스템 기본 세팅

pSystem->createSound("SkyroadsRoad08B.wav", FMOD\_HARDWARE, NULL, &pSound[0]); //배경음악 셋팅

}

void SoundOut() {

pSound[0]->release(); //배경음악 셋팅 해제

pSystem->release(); //사운드 시스템 해제

pSystem->close(); //사운드 시스템 닫기

}

void help() { //콘솔창에 출력될 조작법

cout << "----------------------------------------------" << endl;

cout << " → & ← : 우주선을 좌, 우로 이동시킵니다." << endl;

cout << " ↑ & ↓ : 우주선의 속도를 조절합니다." << endl;

cout << " space : 우주선을 점프시킵니다." << endl;

cout << "----------------------------------------------" << endl;

cout << " s : 배경음을 켜거나 끌 수 있습니다." << endl;

cout << " r : 게임을 리셋합니다." << endl;

cout << " esc : 게임을 종료합니다." << endl;

cout << "----------------------------------------------" << endl;

}

int main(int argc, char\*\* argv) {

help();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH);

glutInitWindowSize(width, height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Skyroads");

SoundIn();

LoadGLTextures();

MyInit();

glutDisplayFunc(MyDisplay);

glutReshapeFunc(MyReshape);

glutKeyboardFunc(MyKeyboard);

glutSpecialFunc(MySpecial);

glutTimerFunc(150 - speed, MyTimer, 1);

glutTimerFunc(2000, jumpTimer, 1);

MenuFunc();

glutMainLoop();

SoundOut();

return 0;

}

참고한 자료

FMOD를 활용한 게임 사운드 - <http://blog.naver.com/ktmgame/20204061271>

교재 코드 – 3-19(Polar View를 이용한 극 좌표계), 4-8(One Window Multi Viewport), 6-5(Multi Texture Mapping), 7-3(Projection Shadows), 9-1(Skybox)