2021년도 프로그래밍방법론및실습 프로젝트 최종보고서

[Rider]

제출일자: 2021. 12. 21

팀원 1: 박정현

팀원 2: 이가은

목 차

1. 프로젝트 개요 (약 2페이지)

가. 프로젝트 소개

○ rider라는 모바일 게임을 모티브로 하여 오토바이가 험난한 지형지물위를 전복되거나 추락하지 않으면서 계속 주행해나가는 게임을 제작한다.

○ 게임의 목표는 최대한 멀리 가므로써 높은 점수를 얻는 것이다. 점수는 일정 거리마다 1점씩 얻을 수 있으며 공중에서 회전에 성공할 경우 추가 점수를 얻는다.

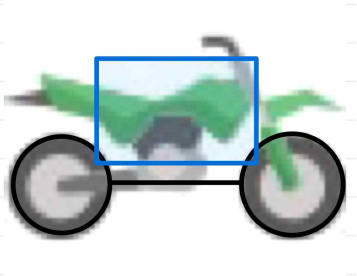
○ 오토바이가 전복되거나 바퀴가 아닌 부분이 지형지물에 닿으면 게임은 즉시 종료된다. 또 바닥으로 추락할 경우에도 게임은 종료된다.

○ 게임에서 오토바이의 움직임을 구현하기 위해서는 물리 법칙에 대한 계산이 들어가야 한다.

나. 개발 범위 및 내용

○ (1) 지형지물과의 충돌

○ 게임 속에서 오토바이의 움직임을 물리 법칙을 적용하여 구현하기 위해서는 게임상에서 지형지물과의 충돌을 계산하는 것이 중요하다. 오토바이는 위치, 속도, 가속도, 회전각, 각속도, 각가속도 등을 가진다. 이들은 중력과 지형지물과의 충돌에 의해 값이 변화하게 된다. 이 게임에서는 오토바이의 바퀴 외 다른 부분이 닿는 경우 게임이 멈추기 때문에 오토바이를 두 바퀴의 결합으로만 단순화하여 물리 법칙을 적용했다.



○ 위 사진에서 보이는 두 원은 오토바이의 바퀴를 표현하고, 직사각형은 오토바이의 몸체로 이 부분이 지형지물과 충돌할 시 게임이 중단되도록 하였다.

○ (2) 사용자 조작의 구현

○ 사용자가 스페이스바를 누르면 가속하고, 떼면 속도를 감속하도록 제작한다. 그리고 오토바이가 공중에서 있을 때 스페이스 바를 누르면 각속도가 증가하여 회전할 수 있도록 한다.

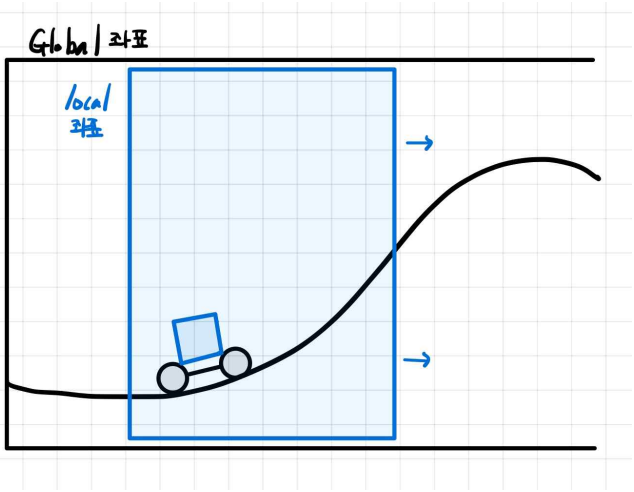
○ (3) 지형지물의 구현

○ 다양한 지형지물을 나타낼 수 있고 미분이 용이하여 기울기를 구하기 쉬운 다항함수의 도입을 생각해냈다. 0차, 1차, 2차 함수와 빈 공간이 있는

ground 함수를 만들었고, init.c에서 각 다항함수의 위치와 계수를 입력하여 반복 루프를 수행할 때 다양한 지형지물이 나타날 수 있게 구현했다. 다이내믹한 게임을 위해 중간에 지형지물의 높이의 변화를 주었고, 모든 함수를 적절하게 섞어 난이도를 조절했다.

○ (4) 화면의 이동

○ 이번 게임은 오토바이가 좌로 길게 움직이면서 이동하므로 화면이 오토바이와 함께 움직여야 했다. 따라서 오토바이가 실제 동작하는 global 좌표와 화면이 보여지는 local 좌표를 구분하였다.



○ 그리고 화면이 움직인다면 오토바이가 자연스럽게 구동하는지 알 수 없기때문에 전처리기 if 문을 사용하여 디버그모드를 따로 만들었다. 디버그 모드로 빌드시 global 좌표계로 화면이 고정된 형태로출력된다.

다. 개발 목표

○ 프로젝트의 개발 목표에 대해 세부 사항별로 목표 설정

표 1 개발 목표

평가 항목	개발 목표	우선 순위
물리법칙을 따른 환경구현	자연스럽게 오토바이가 구동할 수 있는 환경을 구현한다.	1
오토바이 조작	오토바이의 속도가 너무 빠르거나 느리지 않도록 적당한 제약을 준다	2
움직이는 화면 구현	가로로 움직이는 게임에 맞도록 화면이 움직이도록 구현	3
Rider와 비슷가	Rider게임과 최대한 비슷하도록 꾸며본다	4

2. 세부 개발 내용 (약 5페이지)

가. 개발 과정 및 추진 체계

○ 프로젝트 기한 내 세부 수행 계획 및 팀원별 역할 분담 명시

표 2 팀원별 역할 분담

팀원	역할
박정현	주요 코드 작성 및 렌더링 담당
이가은	동역학 원리 수식 검증, 디버깅 담당

표 3 프로젝트 세부 수행 계획

세부 개발 내용	담당자	개발 기간								비고
		11월				12월				
		1주	2주	3주	4주	1주	2주	3주	4주	
sdl2공부	박정현	o	o							
동역학 수식 검토	이가은	o	o							
코드로 구현	박정현					o	o	o		
물리법칙 디버깅	이가은					o	o	o		

나. 빌드 환경 및 플랫폼

○ 보고서를 보고 별첨 소스코드를 통해 빌드가 가능하도록 정확한 개발 환경과 플랫폼을 명시

표 4 프로젝트 개발 환경

항목	사양
운영 체제	*Ubuntu 18.04*
개발 언어	*C*
컴파일러	*gcc*
사용 라이브러리	*SDL2*

○ 빌드 방법 설명(제시한 개발 환경에서 어떤 방법으로 빌드하는지 사진 혹은 스크립트로 제시)

```
# 프로젝트 빌드
$ cd ./src
$ make
$ cd ..
$ ./Rider
```

그림 3 소스코드 빌드 스크립트

다. 소스코드 구조

표 5 소스코드 트리 구조 *(아래 내용은 모두 예시임)*

폴더 / 파일		설명
final		*컴파일 이후 실행 파일이 저장될 폴더*
src	*main.c,main.h*	*main loop를 구현한 파일과 헤더파일*
	action.c, action.h	*실제 게임이 진행되는 부분을 담은 파일과 헤더파일*
	init.c,init.h	*SDL2과 구조체 변수들을 설정하고 초기화하는 함수를 구현한 파일과 헤더파일*
	init.h	*init.c에 대한 헤더 파일*
	input.c	*키보드 입출력 관련 함수를 구현한 파일*
	input.h	*input.c에 대한 헤더 파일*
	draw.c	*그래픽 렌더링 관련 함수를 구현한 파일*
	render.h	*render.c에 대한 헤더 파일*
	utils.h,utils,c	*기타 필요한 구조체 및 열거형 변수들을 선언한 헤더 파일*
	makefile	*프로젝트 컴파일을 위한 makefile*

라. 주요 알고리즘 및 구현 소개

○ main 함수에서 초기화를 진행한 뒤, 프로그램 무한 루프로 들어간다. 루프 내에서 빈 화면 만들기, 키보드 입력받기, 게임이 진행 중일 경우에 오토바이 이동시키기, 렌더링, 화면 출력, 기다리기 순서로 반복한다.

○ main 함수의 무한 루프 중 ActGame() 함수에서 오토바이를 움직이는데 필요한 모든 연산을 진행한다. ActGame()함수에서 오토바이의 현재 위치를 계산하는 순서는 다음과 같다.

○ 1. physics() ; 두 바퀴에 대해 중력, 충격량 등의 물리 법칙을 적용하여 각각 속도를 계산한다.

○ 2. ActBike() : 사용자가 지면에 닿아있을 때 입력을 주면 뒷바퀴의 속도를 증가시킨다. 지면에 닿아있지 않으면 각속도 값을 증가시킨다.

○ 3. UpdatePose() : 만약 2번에서 부여된 각속도가 증가한다면 각속도를 두 바퀴 각각의 속도로 변환시킨다.

○ 4. UpdatePose() : 두 바퀴는 1,2,3번의 연산에 의해 하나의 강체로 움직이기에 적합하지 않은 속도를 가지고 있다. 따라서 이를 하나의 강체에 적합한 속도가 되도록 조정한다.

○ 5. UpdatePose() : 두 바퀴를 속도를 토대로 두 바퀴의 위치를 결정하고 오토바이 중심의 위치 및 각도를 결정한다.

○ 6. UpdatePose() : 바퀴의 위치에 대해 오차 누적되에 차이가 커질 수 있기 때문에 오토바이 중심과 각도를 통해 바퀴의 최종 위치를 다시 계산한다.

○ 1. physics()

○ 물리법칙에 의한 속도의 변화를 계산하기 위해서는 우선 지형지물과의 충돌을 검증해야한다. closesFeature()함수를 통해 지형지물은 flist라는 구조체 포인터 배열에 x좌표의 순서대로 저장되어 있다. 따라서 이를 순서대로 불러오며 가장 가까운 지형지물의 인덱스를 찾는다.

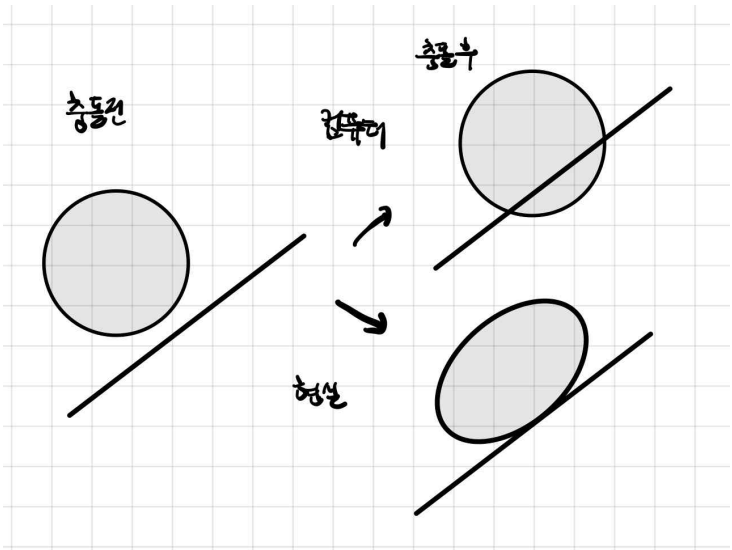
○ 가장 가까운 지형지물과 그 앞 뒤에 대해 충돌 검사를 진행한다. 충돌검사는 우선 충돌을 하는지 않하는지에 대한 여부를 먼저 검사한다. 만약 충동을 하면 어느 정도를 침범하였는지와 충돌면에대한 법선벡터를 저장한다.

○ (1) 충돌여부 검사 detect_contact()

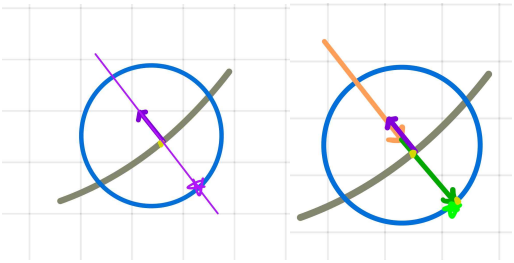
○ 두 바퀴에 대해서는 중심부와 지형지물의 모든 점 사이의 거리를 계산하여 바퀴의 반지름보다 가까워지지 않는지 검사한다. 만약 가까우면 cal_crash_tire()함수를 오출한다. 오토바이의 몸체에 대해서는 지형지물의 x,y값이 몸체의 최소 최대인지만 검사하고 만약 충돌 시 게임을 종료한다.

○ (2) cal_crash_tire() 얼마나 충돌했는지와 접촉면의 법선벡터를 계산

○ 코드상에서의 동역학을 구현할 때 주의해야 할 점은 물체의 이동이 이산적이라는 점이다. 실제에서는 충돌 시 물체의 표면끼리만 만나게 되지만 프로그램상에서는 물체가 겹쳐버리게 된다. 따라서 충돌을 검사할 때 얼마나 침범



했는지도 계산하여 물체의 위치를 서로 침범하지 않도록 조정해 주어야 한다. 우선 침범의 깊이를 구하기 앞서 충돌 부분의 법선벡터를 구했다. 지형지물이 다항함수로 구성하여 때문에 기울기를 쉽게 구할 수 있었고 이를 통해 법선벡터를 구했다. cal_circle_y()를 통해 원의 중심과 가장 가까운 점과 법선 벡터의 교점을 구해서 가장 깊은 점을 구했다.

○ 

○ 그 이후 원의 중심에서 가장 깊은 점까지의 벡터를 구하고 이를 법선벡터와 내적하여 얼마나 침범하였는지 그 깊이를 구했다. 그리고 이를 ActBike.c 내부의 전역변수인 Crash 구조체의 crashs〔〕배열에 저장한다.

○ (3) 중복된 충돌에 대해서 처리

○ 평면 지형지물의 경우 같은 지형이 연속되어 있는데 각각의 지형지물에 대해 계산하다보니 다른 충돌로 계산한다. 이를 법선벡터와 충돌한 바퀴가 같으면 같은 충돌로 처리한다. 그리고 최종적인 충돌을 finalcrash〔〕라는 Crash 구조체에 저장한다.

○ (4) adjust_pose()

○ finalcrash에 저장된 충돌에대해 침범이 일어난 만큼 위치를 재조정해준다. 이때 아에 안 겹치게하면 충돌 여부가 반복적으로 나타나 진동이 발생하므로 GAP만큼 겹치게하여 오토바이의 움직임을 안정시킨다.

○ (5) Force()

○ 충돌 후 결정되는 속도를 계산하여 각 바퀴의 속도값을 변화시킨다.

$$e = \frac{(v_G)_0 - (v_G)_2}{(v_G)_1 - (v_G)_0}$$

○ 반발계수 E(0~1)를 정의하여 법선벡터 방향의 속도는 E만큼 감소된 값으로 설정하고 법선벡터와 수직한 방향의 속도는 값을 감소시키지 않고 그대로 적용한다. 그리고 접선의 기울기를 고려하여 중력가속도만큼 속도를 변화시킨다.

$$e = \frac{\omega_0 - \omega_2}{\omega_1 - \omega_0}$$

○ 2. ActBike()

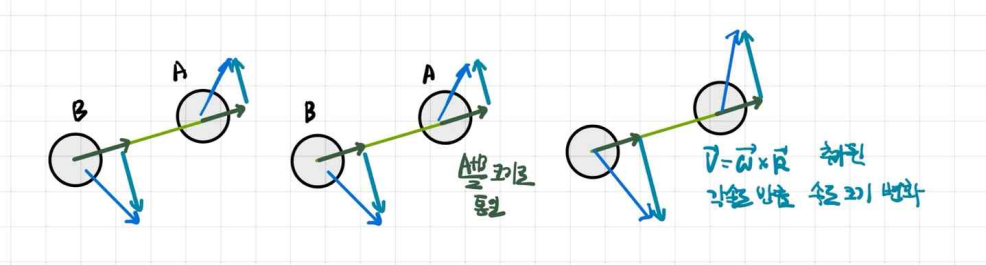
○ 뒷바퀴가 지면에 닿아 있는지 여부를 충돌검사 단계에서 저장한다. 그리고 ActBike에서 뒷바퀴가 닿아있으면 Bike의 기울기에 비례해서 각각 증가시킨다. 뒷바퀴가 지면에서 떨어져 있으면 각속도를 증가시킨다. 만약 스페이스바이 입력되지 않으면 원래대로 속도를 줄이거나 각속도를 감소시킨다.

○ 3. Updatepose()

○ 두 바퀴의 속도를 조정하고 이를통해 오토바이의 위치를 업데이트한다.

○ (1)update_tire_vel()

○ 위 그림의 순서대로 속도의 조정을 실행한다. physics()와 ActBike()에서



계산한 결과는 하나의 강체로써의 운동이 아닌 각각의 타이어의 운동이다. 따라서 우선 AB가 같은 크기의 선속도를 가질 수 있도록 AB상대 위치 벡터와 평행한 속도 벡터의 크기를 평균을 구해 조정한다. 그리고 만약 ActBike()에서 추가적인 각속도가 부여가 되었다면 이를 선속도로 바꾸어서 상대 위치 벡터와 수직한 방향의 벡터로 추가해준다.

○ (2)이후 각 바퀴를 각 속도만큼 위치를 변화시킨다.

○ (3)update_Center_theta()

○ 오토바이가 지금 회전해 있는 각도를 구한다.

○ (4) Update_Body()

○ 오토바이의 회전각과 바퀴의 위치의 중점을 통해 오토바이가 어떤 위치에 있는지 구한다.

○ main 무한 루프의 DrawGame()함수

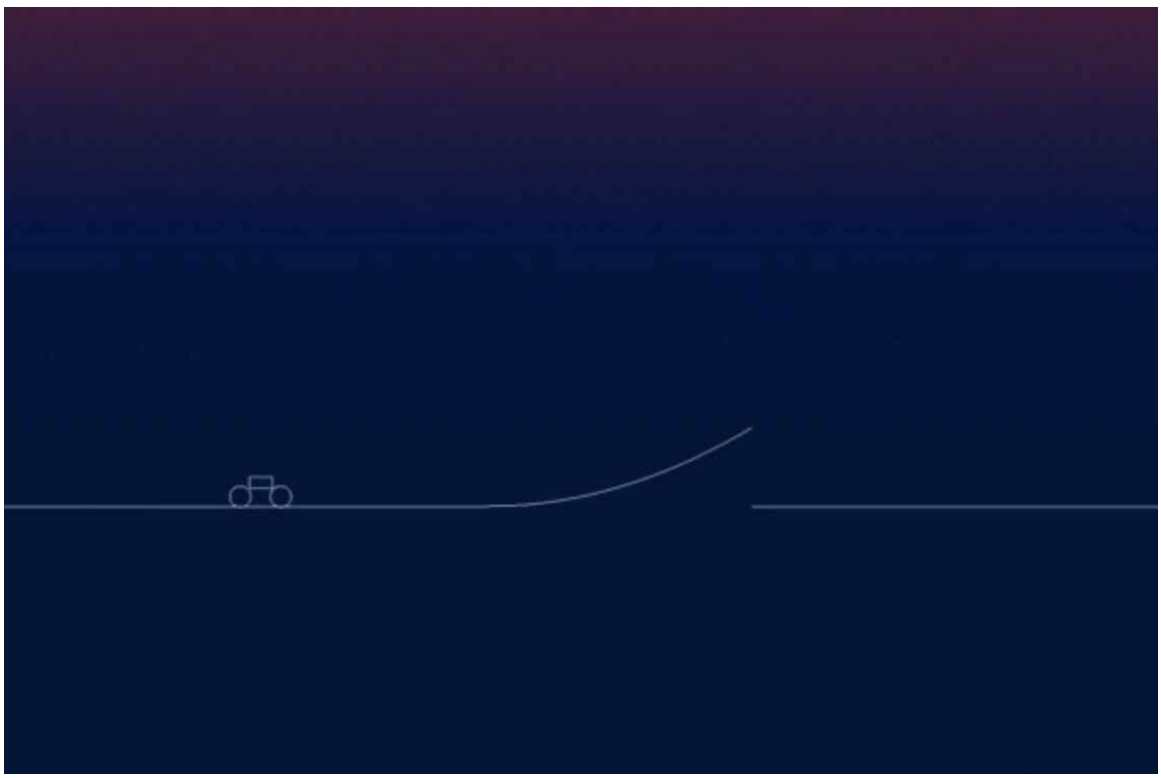
○ DEBUG모드이면 실제 충돌을 판단하는 경계선을 출력하고 고정된 화면으로 출력한다. DEBUG모드가 아니면 오토바이의 이미지를 렌더링하고 움직이는 화면으로 출력한다.(DEBUG는 def,h에서 변경 할 수 있다)

○ 움직이는 화면은 오토바이가 x=300에 항상 위치하도록 global x좌표에서 (global.x - 300)만큼의 값을 뺀 뒤에 렌더링한다.

3. 동작 설명 (약 2페이지)

가. 프로그램 동작 화면

DEBUG모드



나. 개발 목표 달성도

○ 1. 다.에서 설정한 개발 목표 대비 달성도를 평가

표 6 개발 목표 달성도

평가 항목	개발 목표 대비 달성도	우선 순위
물리법칙을 따른 환경구현	겉보기에는 자연스러운 움직임을 하지만 중간에 값이 튀거나 하는 부분이 있고 자연스러운 움직임을 위해 실제 물리법칙과 동떨어진 계산을 하는 부분들이 있다	1
오토바이 조작	스페이스바를 멈추면 각속도가 0이 되어야하는데 그렇치 않다. 그러나 나머지 조작은 잘되는것같다	2
움직이는 화면 구현	목표한 바를 이루었다	3
Rider와 비슷한가	디자인이 조금 아쉽다.	4

4. 결론

○ 이 프로젝트를 시작했을 때 물리엔진이 필요한지 모르고 시작했습니다. 하지만 진행하며 물리엔진이 필요하다는 것을 알게 되었고, 오픈된 물리엔진 라이브러리를 사용하려고 하였지만 찾을 수 없었습니다. 따라서 직접 비슷하게라도 직접 구현해야 했고 많은 시간이 걸렸습니다. 따라서 저희는 프로젝트를 시작하기전 좀 더 많은 사전조사이후 시작해야겠다고 생각했습니다.

○ 프로젝트를 진행하면서 다양한 구조체의 활용, 여러 파일에서의 전역변수의 사용 등을 경험해보며 C언어와 프로그래밍에 대한 경험이 늘었습니다.

5. 참고자료

[1] 2d 물리엔진 구현을 위한 글
Available online: https://gall.dcinside.com/mgallery/board/view/?id=game_dev&no=17921 (accessed on 2021-12-21).