CSED451 Assignment #3-1

3D DRAWING REPORT

Team: GPS

CSE 20202405 문민재 csmmj4594

CSE 20202728 김진수 fusion4268

**Overview of Program**

본 프로그램은 기존 assn1, 2에서 구현한 2D game을 확장하여 3D로 표현하고, 3인칭 시점과 1인칭 시점 두 가지 시점의 perspective view를 구현하였다. 게임의 기본적인 기능은 지난 assn2에서 구현한 것과 동일하지만, 캐릭터와 벽과 같은 오브젝트의 경우 2D에서 3D로 옮겨주면서 좌표를 확장하거나, 기본 도형을 2D에서 3D 도형으로 변경하거나 하는 등의 변화가 있었다. Assn2로부터 확장된 기능은 크게 3D 오브젝트 구현, 3D 시점 구현, 그리고 시점 변화 기능이 있다.

3D 오브젝트의 구현은 기존 2D 오브젝트에서 z좌표를 추가하여 구현하였다. 그에 따라 캐릭터와 벽도 입체적 모양을 나타내야 했는데, 3D 캐릭터와 벽 모양의 구현에 필요한 기본적인 3D 도형은 glut에서 기본적으로 제공하는 함수들을 이용하여 그렸다. 이번 assn3에서는 벽이 도둑과 부딪히면 색뿐만 아니라 모양도 변화해야 했는데, 벽의 5가지 모양들 또한 glut에서 기본적으로 제공하는 함수들과 우리가 자체적으로 구현한 drawCuboid 함수를 이용하여 구현하였다.

3D 시점 구현은 openGL의 matrix mode를 Projection mode와 Modelview mode를 모두 사용하고, Projection mode에서 glPerspectiveView, Modelview mode에서 gluLookAt 함수를 사용하여 3D 시점을 구현하였다.

시점 변화 기능은 키보드의 숫자 키를 눌러 사용할 수 있으며, ‘1’을 눌러 1인칭 시점을, ‘3’을 눌러 3인칭 시점을, 그리고 ‘9’를 눌러 assn2와 같은 XY평면을 바라보는 시점을 적용할 수 있다. 이 기능은 각 시점에 해당하는 카메라 변수들을 미리 선언해둔 뒤, 키보드 입력을 받으면 gluLookAt 함수에 각 시점에 해당하는 카메라를 적용시켜 시점 변화 기능을 구현하였다.

**Programming Environment**

Visual Studio 2019

OpenGL – 4.6

GLSL – 4.60

Freeglut-MSVC-3.0.0

Glew-2.1.0

Glm-0.9.9.7

**Design and Implementation**

1. 자료구조 설계
2. Color (colors.h)

게임 상에서 사용할 색상 코드와 팔레트 변경을 담당한다.

1. Object (object.h)

맵 상의 오브젝트를 정의하는 클래스이다.  
멤버 변수로 type, color, x, y, z를 가지고 있다. 이들은 각각 오브젝트 타입(0일 경우 wall, 1일 경우 character), 오브젝트의 색깔, 오브젝트를 그릴 때 필요한 x, y, z좌표에 해당한다.

1. Wall: public object (Wall.h)

Object를 상속받아 맵 상의 벽을 정의하는 클래스이다.  
Wall 오브젝트들은 공통적으로 type이 0이며, x, y, z는 벽의 위치를 의미한다. 추가적으로 width, height, depth라는 멤버 변수를 가지는데, 이는 각각 직사각형의 x, y, z축 방향 길이를 의미한다.

1. Character: public object (character.h)

Object를 상속받아 맵 상의 캐릭터 오브젝트(플레이어, 도둑)를 정의하는 클래스이다.  
character 오브젝트들은 type이 1에 해당하며, x, y, z는 torso의 중심 좌표를 나타낸다. 그리고 원의 반지름을 나타내기 위한 변수인 rad와 벽과의 충돌 여부를 나타내는 변수인 isCollided가 있다. 그 외에 인간형 캐릭터 구현에 필요한 여러 멤버변수들을 가지고 있다.

1. treeNode (treeNode.h)

인간형 캐릭터의 hierarchy를 표현하기 위해 필요한 Left Child Right Sibling(LCRS) tree를 구현하는 클래스이다.

현재 modelView matrix의 오른쪽에 곱해질 기본적으로 matrix인 mtx,  
기본 mtx transform에 추가적으로 행해질 transform을 나타내는 additionalTransform,  
이 node에 해당하는 신체 부위를 그려주기 위한 함수 포인터인 draw,  
그리고 sibling과 child 정보를 가지고 있다.

1. pose (pose.h)

플레이어와 도둑의 포즈를 정의한 클래스이다.  
캐릭터의 포즈에 따른 변화는 상체에서만 이루어지므로, 왼팔과 오른팔 각각 2가지씩 총 4가지 각도와 포즈에 따른 색을 변수로 갖고있다.

1. camera(camera.h)

이번 과제에서는 3가지 시점을 구현했는데, 각 시점에 해당하는 카메라 정보(camera position, reference point, up vector))을 저장하는 클래스이다.

1. 기본 기능 구현(이전 ASSN 관련)
2. 벽의 생성 및 이동

벽의 생성과 이동은 매 frame(1/60초)마다 호출되는 moveWall() 함수에서 이루어진다. 벽의 형태가 2D에서 3D로 변화함에 따라 벽의 색만 변경하던 것에서 색과 모양을 동시에 바꿔주거나 재생성 위치가 3D 좌표로 달라진 것 외에는 assn2와 동일하다.

1. 충돌 판정

Assn2와 동일하게 진행되었다.

1. 플레이어 캐릭터의 조작

Assn2와 동일하게 구현되었다.

1. 치트 기능

Assn2와 동일하게 구현되었다.

1. Pass/Fail 시의 action
   1. Pass

2D에서 3D로 시점이 변경됨에 따라 카메라의 줌인 및 남은 life의 출력 위치 고정을 여기에서 담당하지 않게 되었다. 그 이외에 기본적인 pass 시 행동(Pass 체크 로직, 벽 속도 증가, 도둑의 포즈 전환 간격 감소, 캐릭터의 전진 등)은 assn2와 동일하다.

* 1. Fail

Assn2와 동일하게 구현되었다.

1. Jump 구현

Assn2와 동일하게 구현되었다.

1. 3D 캐릭터 구현

2D에서 3D로 변경됨에 따라  
head: 원->구, torso: 직사각형 -> Cylinder, limb: 원+직사각형 -> 구+cylinder  
로 변경되고, 2D좌표를 3D좌표로 변경한 것 이외에는 인간형 캐릭터 구현에 필요한 아이디어(LCRS tree 및 hierarchy 구조, pose, 상/하체 애니메이션 등)과 실제 구현은 assn2와 동일하다.

그 외에도, 기존 assn2에서 2D에서 캐릭터를 그려내야 했기 때문에 상체는 정면을, 하체는 측면을 그려내는 이집트 벽화 같은 느낌으로 그려냈지만, 이번 assn3에서는 3D로 캐릭터를 그려내기에 실제 사람과 같은 모습으로 그려냈다.

1. 3D 벽 구현

기존 assn1,2에서 벽 오브젝트를 구현할 때 사용했던 ‘rect’ 객체 대신 벽을 위한 새로운 클래스인 ‘wall’을 정의하였다.

이번 과제에서 도둑이 벽에 충돌하면 벽의 색과 더불어 모양도 바뀌어야 하는데, Cone, Torus, Reversed T-shape, T-shape, 그리고 기본 직육면체 벽 모양의 총 5가지 모양으로 이를 구현하였다.

Cone 모양과 Torus 모양은 glut 내에 정의되어 있는 glutSolidCone 함수와 glutSolidTorus 함수를 사용했고, Reversed T-shape, T-shape, 그리고 기본 직육면체 모양은 drawCuboid 라는 함수를 직접 정의하여 구현하였다. drawCuboid 함수는 중심의 x, y, z좌표와 width, height, depth를 받아 화면상에 cuboid를 그려주는 함수이다.

도둑과 부딪히기 전에는 회색의 기본 cuboid 모양의 벽이었다가, 도둑과 부딪히면 도둑의 색/포즈에 따라 벽의 색/모양이 함께 변경된다. Red – Cone, Green – Torus, Blue – Reversed T, Yellow – T로 매핑된다.

1. 3D 시점 구현

3D 시점의 구현은 display 함수인 display3D() 내에서 이루어진다.

먼저, Matrix mode를 GL\_PROJECTION으로 변경한 뒤, Identity로 초기화 시켜주고, gluPerspective 함수로 perspective projection의 3D 시점을 구현하였다. 이 때, gluPerspective에 필요한 화면 종횡비는 reshape 함수의 argument인 width와 height를 통해 얻어온다.

그리고, Matrix mode를 GL\_MODELVIEW로 바꾼 후, identity로 초기화 시켜주고 바로 gluLookAt 함수를 호출하여 카메라의 위치, 시야 방향, view up vector를 설정해 담고 싶은 화면 각도를 담아낸다.

그 이후에 화면상 그려지는 오브젝트들(wall, player, thief 등)을 화면 상에 그린다.

1. 시점 전환 및 줌 인/아웃
2. 시점 전환

이번 assn3-1에서는 총 3가지의 시점을 구현하였다.  
첫째, 카메라가 플레이어 눈 앞에 있는 1인칭 시점  
둘째, 카메라가 플레이어 오른쪽 어깨 뒤 어딘가에 있는 3인칭 시점  
셋째, XY평면을 비추어 assn2처럼 보이게 하는 3인칭 시점

이러한 세 개의 시점의 전환은 키보드 숫자 키로 이루어지며, 키보드 ‘1’을 입력하면 1인칭, ‘3’을 입력하면 3인칭 시점, ‘9’를 입력하면 XY평면을 보는 시점으로 전환된다.

시점 전환의 구현은 이 3가지 시점에 해당하는 카메라 정보(eye, lookat, up vector)들을 미리 정의해 두고, keyboard function에서 누르는 숫자에 따라 cameraMode라는 변수를 변경하게 했다. 매 프레임마다 호출되는 함수인 frameAction에서 cameraMode에 따라 알맞은 카메라 정보를 가져오고, 마지막으로 이 카메라 정보를 display() 함수 내의 gluLookAt에 전달하는 방식으로 시점 전환을 구현하였다.

1. 줌 인/아웃

Assn1, 2와 동일하게, 캐릭터가 벽을 점프하지 않고 성공적으로 통과할 때마다 화면이 줌되어 더 좁은 영역을 비춰야 한다.

Assn1, 2에서는 gluOrtho2D 함수의 시작과 끝 x, y좌표를 변경하여 줌인을 구현하였다면, 이번 assn3-1에서는 display 및 reshape 함수에서 호출되는 gluPerspective 함수에서의 fovy(시야각) 값을 변경하여 줌인을 구현하였다. Fovy 값을 조절하여 더 좁은 시야각을 가지게 한다면, 3인칭 시점에서 플레이어 캐릭터의 이동에 따라 카메라의 위치를 이동시키지 않아도 마치 카메라가 줌 인 되는 듯한 화면을 구성할 수 있다.

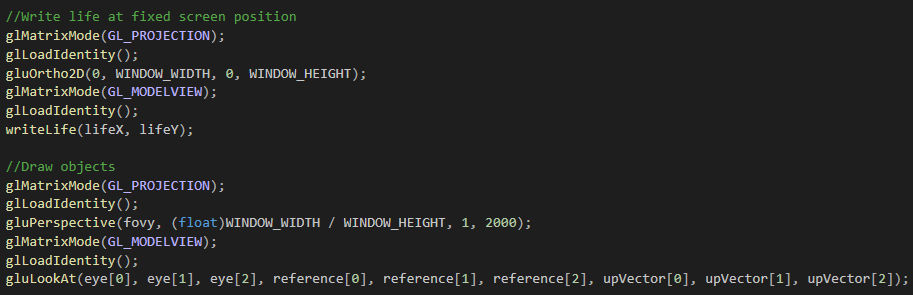
플레이어가 점프 없이 벽을 성공적으로 통과한다면, fovy를 기존의 0.85배로 설정하고, 자연스러운 줌을 위해 zoomFrame 기간동안 천천히 줌 될 수 있도록 fovyPerFrame을 설정하였다.

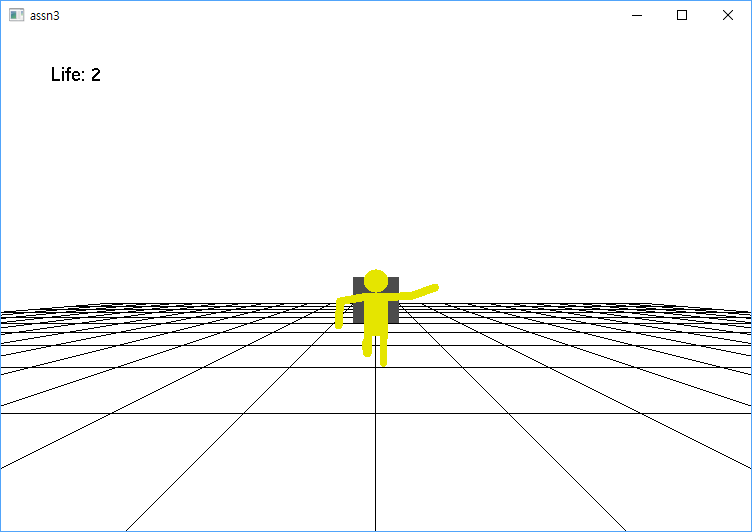
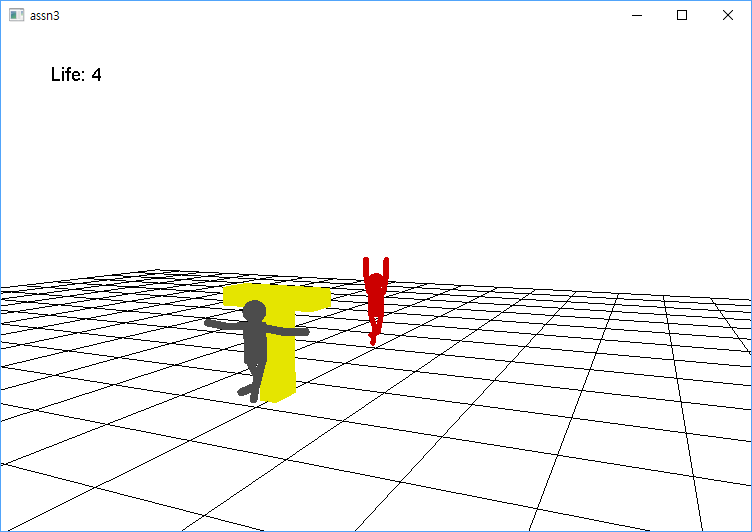
도둑이 점프하지 않았는데 플레이어가 점프하여 벽을 통과할 경우, 플레이어 캐릭터만 뒤로 이동하므로 플레이어가 화면에서 사라지는 것을 방지하기 위해 fovy를 기존의 1.05배로 설정하여 world상의 더 넓은 영억을 비출 수 있게 설정하였다.

1. 화면 상 고정된 위치에 life 출력(추가 기능)

플레이의 편의를 위해 화면 상 고정된 위치(왼쪽 상단)에 현재 남은 life를 출력해준다.

이 Life 출력은 world 위가 아니라, device 좌표계 위에서 이루어지고, 또한 world 위에서 출력해주는 object들 및 1인칭/3인칭 뷰의 시점과 별개로 이루어져야 한다. 따라서, display 함수 내에서 오브젝트들을 그려주는 것과 별개로 GL\_PROJECTION matrix mode로 진입하고, gluOrtho2D 함수를 이용해서 화면 상에 life를 출력해주었다.





**How to Run**

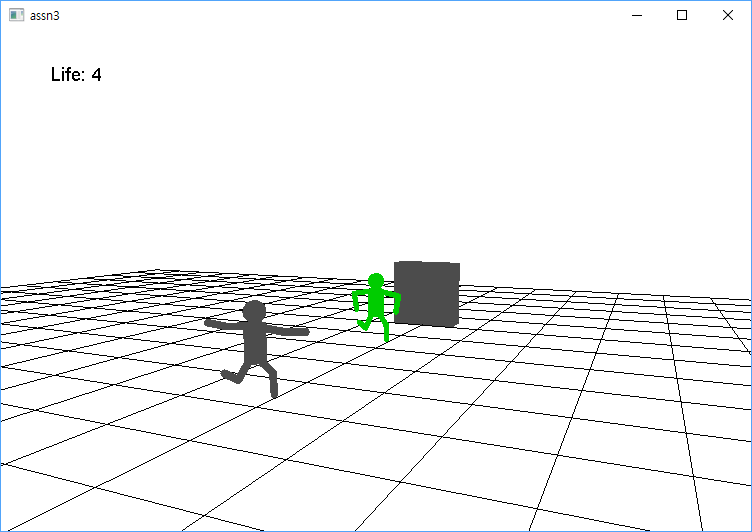
프로젝트 폴더의 \bin\x64 폴더로 이동하여 graphics\_assn3.exe를 실행시킨다.

캐릭터의 포즈 변경은 방향키로 입력하며, 위, 아래, 왼쪽, 오른쪽 방향키는 각각 플레이어 캐릭터를 적색, 녹색, 청색, 황색으로 변경시키고, 플레이어 캐릭터는 각 색에 대응하는 포즈를 취한다. 스페이스바를 입력하여 캐릭터를 점프시킬 수 있다.

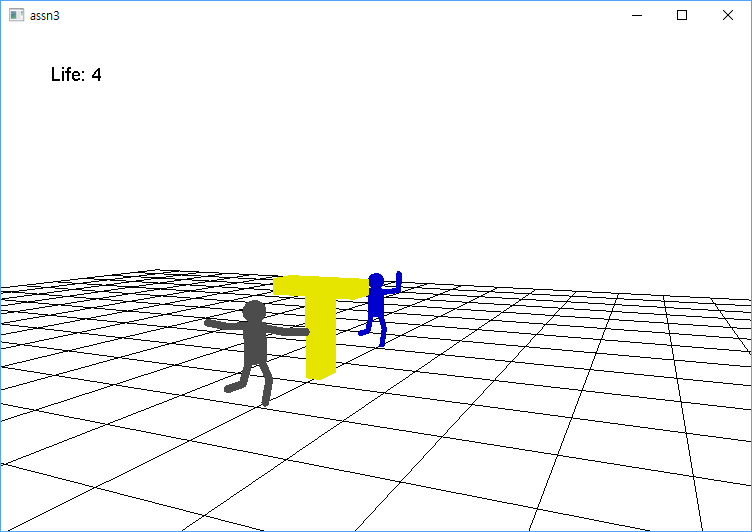
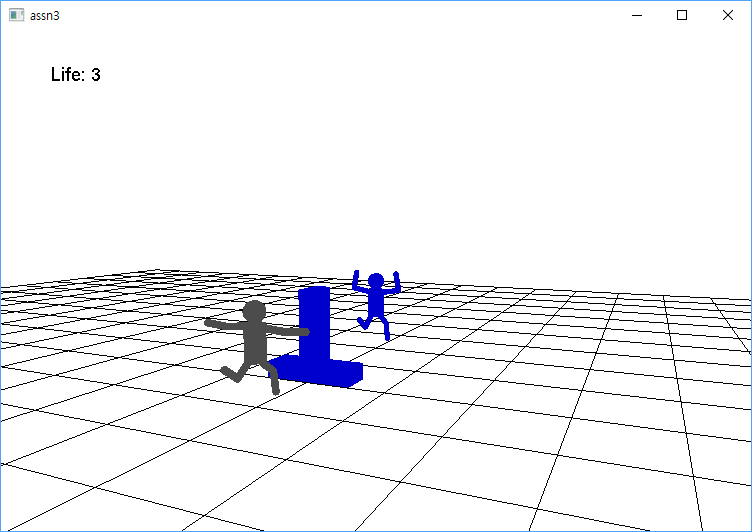
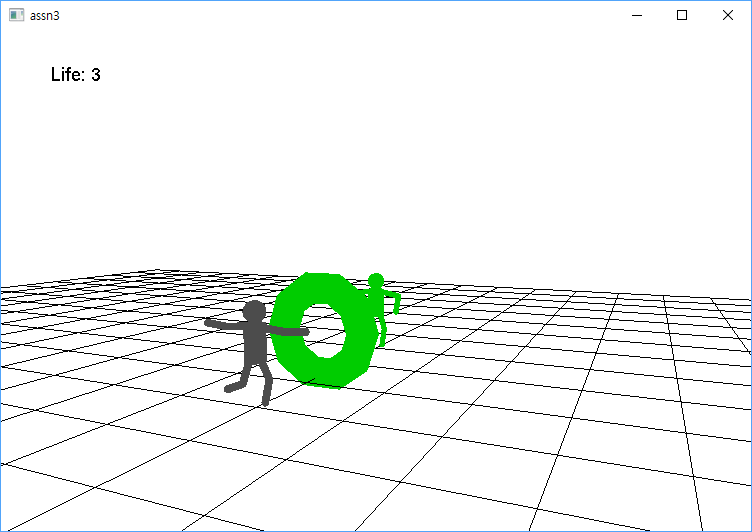
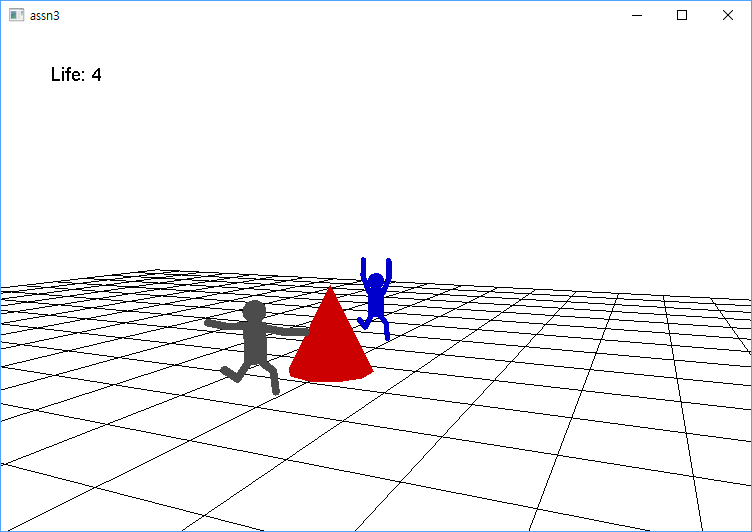
숫자키 1번을 눌러 1인칭 시점으로, 3을 눌러 3인칭 시점으로, 9를 눌러 XY 평면을 바라보는 시점으로 시점을 전환시킬 수 있다.

**Example**

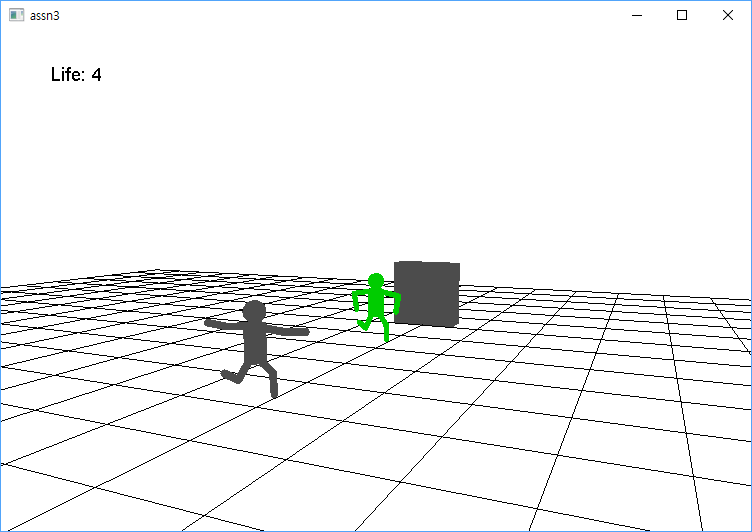
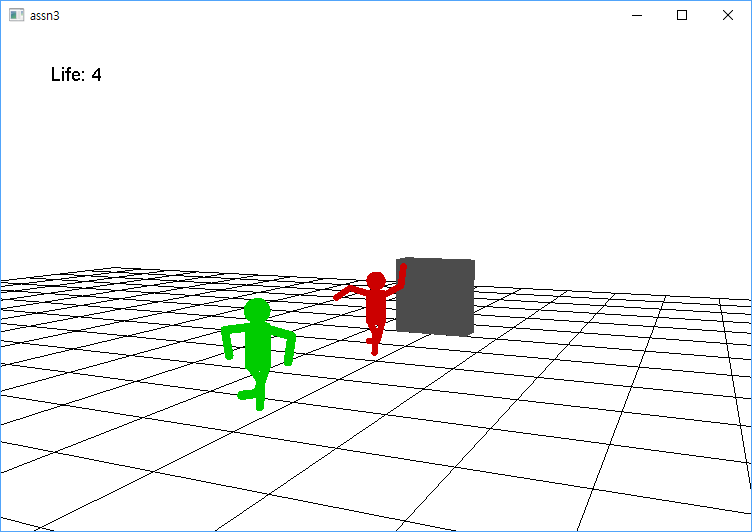
기존 Assn2와 중복되는 부분은 제외하였다.



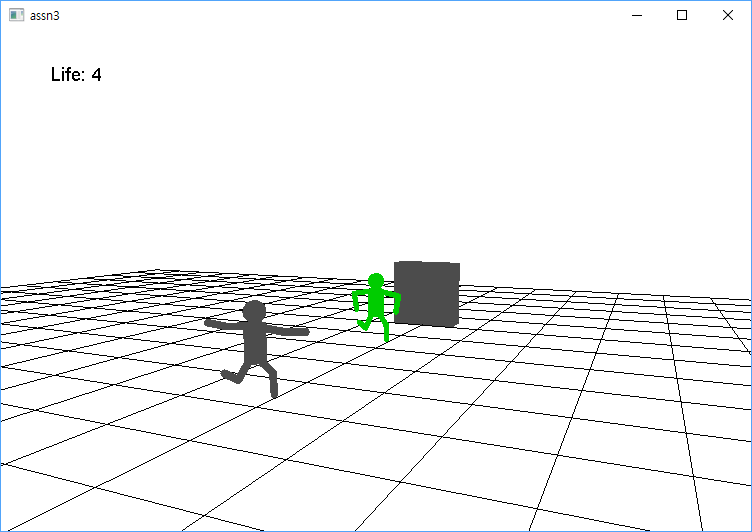
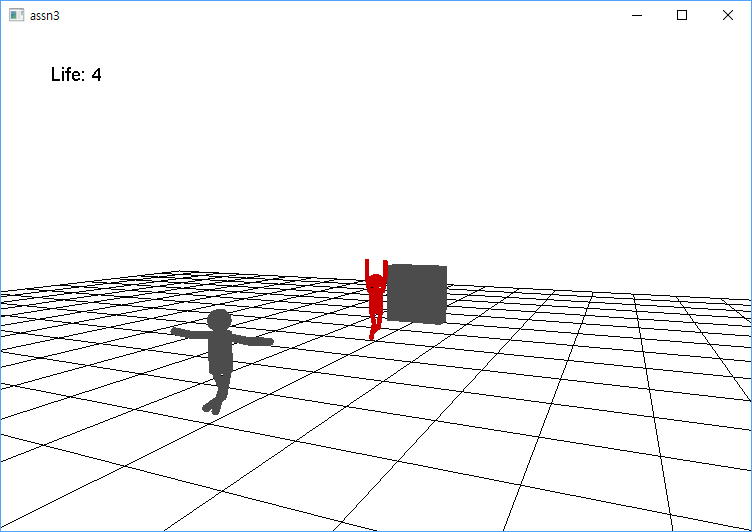
프로그램 실행 시 화면으로, 좌측 상단에 현재 life가 표시되고, 3인칭 뷰로 시작한다. 벽은 기본적으로 회색 직육면체 모양이다.



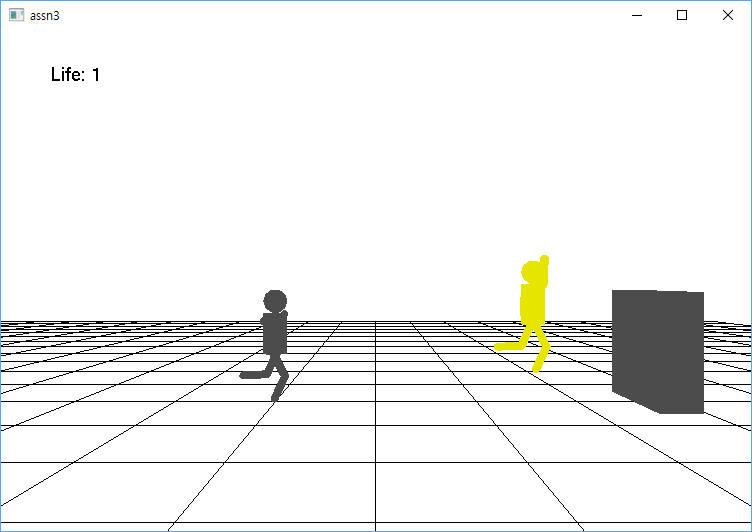
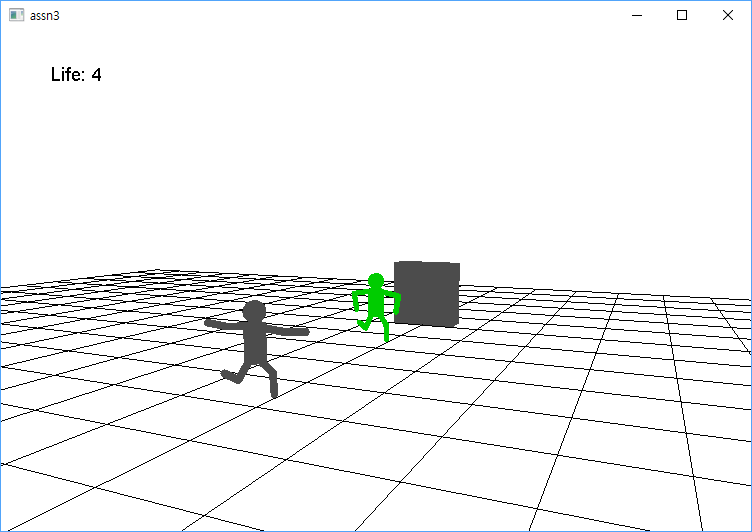
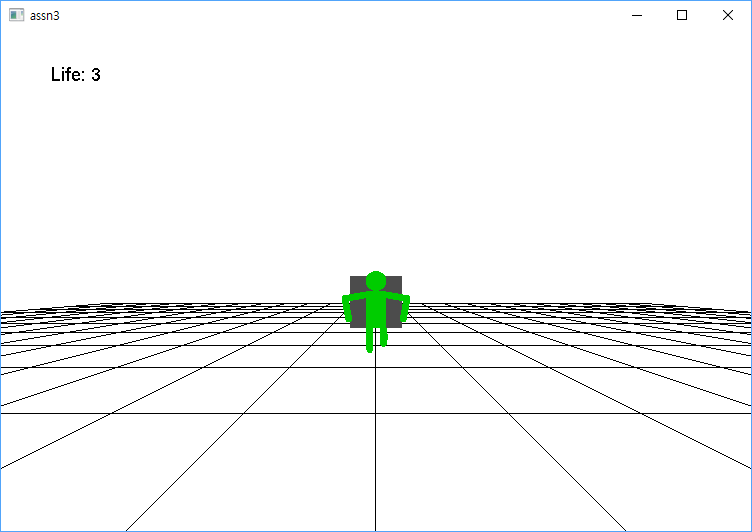
도둑의 색과 자세는 일정시간마다 랜덤하게 변경되며, 도둑 색의 변경 간격은 콘솔에 Pass가 출력되는 횟수가 많아질수록 줄어든다.  
도둑과 벽이 충돌할 경우 벽의 색이 도둑의 색과 그 색에 해당하는 모양으로 변경되어 플레이어에게 다가온다. Red – Cone, Green – Torus, Blue – Reversed T, Yellow – T 모양이다.

 🡪 

플레이어가 벽을 점프하지 않고 성공적으로 통과할 경우, 플레이어가 도둑에게 조금 더 다가가고, 시야각이 좁아져 조금 더 좁은 화면을 비추게 된다. 1인칭 뷰, 3인칭 뷰 모두 해당된다.

 🡪 

도둑이 점프하지 않았는데 플레이어만 점프하여 벽을 통과할 경우, 점프로 인해 플레이어가 도둑으로부터 멀어지게 되므로, 시야각을 넓혀 조금 더 넓은 화면을 비추게 된다. 1인칭 뷰, 3인칭 뷰 모두 해당된다.



숫자 키 ‘1’을 눌러 1인칭 뷰로 변경할 수 있고, ‘3’을 눌러 3인칭 뷰로 변경할 수 있다.  
추가적으로 ‘9’를 눌러 XY평면을 보는 뷰로 전환할 수도 있다.  
시점을 전환하여도 화면 왼쪽 상단에 남은 life가 고정되어 출력된다.

**Discussion**

1. 3D 캐릭터/벽 구현
2. 캐릭터 구현

이번 과제의 요구사항이 2D에서 3D로 변화됨에 따라 캐릭터와 벽의 모양도 평면에서 입체로 변경되어야 했다. 그에 따라 캐릭터의 각 부분을 구현 때에 원, 사각형과 같은 기본 도형을 구, 원기둥 등으로 변경해야 했고, 이러한 기본 3D 도형들을 직접 구현하기 위한 방법을 고민했었으나, glut에 glutSolidSphere, glutSolidCylinder 등의 내장된 여러 입체 도형들이 있었고, 이러한 glut에서 제공되는 입체 도형들을 활용하여 3D 캐릭터를 구현하였다.

1. 벽 구현

벽의 구현도 캐릭터와 마찬가지로 3D로 변화함에 따라 벽의 입체 모양을 구현할 때 구현할 모양과 구현할 방법을 고민했었으나, glut에서 제공하는 glutSolidCone, glutSolidTorus 함수와 이번 assn에서 자체적으로 구현한 drawCuboid 함수를 이용하여 Cone, Torus, Reversed-T, T, 그리고 박스 모양의 5가지 모양의 3D 벽을 구현할 수 있었다.

1. 3D 시점 구현
2. 2D 시점에서 3D 시점으로의 변화

이전 assn1, 2에서는 gluOrtho2D를 이용한 2D 시점의 게임을 구현했으나, 이번 assn3에서는 3D 시점을 구현해야 했고, 2D 시점을 3D 시점으로 옮기는 과정에서 시점은 의도한 대로 구현되었으나, 오브젝트의 위치들이 이상하게 배치되어 있는 현상이 발생했었다. 이러한 현상의 원인은 glLoadIdentity() 함수로, 이전 assn2의 구현에서 glLoadIdentity()를 필요하지 않음에도 불구하고 지나치게 많은 곳에 사용했었고, 그 결과 3D로 옮기는 과정에서 glLoadIdentity가 의도치 않게 적용되어 오브젝트들의 배치가 이상해진 것이었다.

이 문제를 해결하기 위해 glLoadIdentity()의 사용을 자제하고, 대신 glPushMatrix와 glPopMatrix를 이용하여 구현한 결과, 3D 시점 및 오브젝트들의 배치가 의도한 대로 이루어졌다.

1. 3D 줌 인

이번 assn3에서도 assn1, 2와 같이 캐릭터가 성공적으로 벽을 통과할 때마다 화면이 줌인되는 기능을 구현해야 했는데, 이러한 줌 인 기능을 구현할 방법에 대해 고민했었다. 처음에는 플레이어 캐릭터의 이동에 따라 카메라의 위치를 변경하여 줌 인을 구현하는 방안을 생각했는데, 1인칭 시점에서는 구현하는 것이 어렵지 않았으나, 3인칭 시점에서 캐릭터가 움직일 때마다 카메라의 위치를 그에 맞게 옮겨주고, 카메라의 각도 또한 재조정하는 것이 어렵다고 생각했었다.

그래서 줌 인을 구현하기 위해 다른 방안으로 생각한 것은 glPerspective의 argument로 들어가는 fovy, 즉 시야각을 조절하는 것이었다. Fovy 값을 낮추어서 시야각을 더 좁게 설정하면 결과적으로 더 좁은 world의 영역을 비추어 줌 인이 되는 것이었다. 결과적으로, Fovy 값 설정을 통해 카메라 위치를 옮기는 것보다 쉬우면서 더 자연스럽게 줌 인을 구현할 수 있었다.

**Conclusion**

1. 3D 시점 구현

이번 assn3을 통해, openGL matrix의 Projection 모드와 modelview 모드의 존재 의의를 더욱 확실히 알게 되었고, Projection 모드와 modelview 모드의 선후 관계와 glPerspective 함수 및 gluLookAt 함수와 오브젝트들을 그려주는 함수들을 어떤 순서로 배치해야 할 지를 이해하게 되었다.

또한, 각 모드에서 glLoadIdentity는 정말 필요한 곳에만 사용하고, 그 외의 경우에는 glPushMatrix와 glPopMatrix를 사용하는 것이 matrix가 꼬이지 않고 안전하다는 것을 알게 되었다.

그 외에도, perspective view에서 glPerspective의 fovy 값 조절을 통해 비추고자 하는 화면을 더 넓게/좁게 설정하여 줌아웃/인을 구현할 수 있음을 알게 되었다.

**Direction of Improvement**

1. 벽의 원근감 부족

현재 구현된 프로그램에서는 벽이 다가올 때의 원근감이 부족하다. 이러한 단점은 특히 1인칭 시점에서 두드러지는데, 벽의 모양만 잡혀있고, 벽의 테두리를 알려주는 외곽선이 없고, 또한 벽의 색이 캐릭터(도둑, 플레이어)의 색과 정확히 일치하기 때문에 이에 파생하여 캐릭터와 벽이 겹치면 잘 보이지 않는 등의 문제가 발생한다.

하지만, 광원을 추가하여 벽과 광원 사이의 관계에 따라 빛이 달라져 색이 조금씩 달라지거나, 벽의 색을 캐릭터의 색과 조금 다른 색으로 설정하거나, 벽에 외곽선을 추가해준다면 벽의 원근감을 조금 더 살릴 수 있을 것으로 보인다.

1. 벽의 갑작스럽고 어색한 모양 변화

현재 구현된 프로그램에서는 도둑과 벽이 충돌하여 벽의 모양이 급작스럽게 바뀌어 부자연스럽다. 특히, 현재 assn3에서는 벽에 캐릭터 모양의 구멍을 뚫는 것이 아닌, 벽의 모양을 임의의 모양으로 바꾸는 것이기 때문에 벽의 모양이 갑작스럽게 변하는 것이 특히 어색하게 느껴진다.

이 문제는 벽의 모양 변화에 애니메이션을 넣어 시간을 두고 조금 더 자연스럽게 벽의 모양이 바뀌게 하거나, 도둑과 벽이 충돌할 때 도둑의 포즈 모양으로 벽에 구멍을 뚫어주는 방식으로 구현한다면 해결될 수 있을 것으로 보인다.

**참고문헌**

<https://alleysark.tistory.com/260>: 이번 과제의 구현에서, 파일로 된 vertex/fragment shader를 string으로 읽어오는 코드(readShaderSource 함수)와,  
program에 vertex/fragment shader를 attach 완료한 후 program을 점검하여 문제가 있을 경우 로그를 띄워주는 CheckProgram 함수를 참고하였다.

<https://heinleinsgame.tistory.com/7>: GLFW에서 shader를 적용시켜 삼각형 하나를 출력하는 기본적인 코드로, 이번 과제에서 shader를 사용하기 위한 setup(VAO, VBO, program setting 등)을 참고하였다.