개요 / How to run / 주의 사항

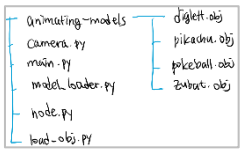

그림 1 필수적인 파일 트리, 반드시 해당 파일 트리대로 구성된 채로 실행되어야 합니다.

[구현 사항] 1. project 1에서 사용된 카메라, grid plane(xz), frame(xyz)
2. single mesh 렌더링: z키로 wireframe(default)/solid 전환 가능, 드래그 앤 드랍으로 obj 파일을 볼 수 있음, light source 3개 구현, 새로운 obj 파일 드랍하면 그것만 그림, vertex position과 vertex normal, face information 파싱해서 그리기, 1-B-iv의 정보 출력, animating 상태에서 obj 파일 드랍시 single mesh 렌더링으로 돌아오기 등.
3. H 키를 눌러서 animation 모드 진입, node tree를 기반으로 부모의 transformation에 의존하는 계층 모델 애니메이션 렌더링(높이 3의 이진 트리 구조)
4. 모든 오브젝트를 phong illumination과 phong shading 기반으로 그리기, 여러 개의 light source(모두 point light) 사용, wireframe/solid 전환 기능(z) 등.
Submission.zip 파일 압축을 푼 상태에서, 폴더 내부 구조를 변경하거나 일부 코드 파일이 없으면 안됩니다. 즉, 그림 1과 같은 파일 트리 구조를 가져야 합니다. 또한, 과제 명세대로 os 모듈을 추가적으로 import하여 사용합니다.
`> python main.py` 로 실행할 수 있습니다(python main.py가 디렉토리 안에서).

Implements: Manipulate camera

Project1에서 제출한 카메라와 코드는 비슷합니다. 다만, Project2에서 코드가 대량 추가되었기 때문에, camera.py라는 파일로 코드를 빼서 class 형태로 main에서 import하도록 바꾸었습니다. 또한, 작은 grid plane을 그립니다.

Implements: Single mesh rendering mode

1. main.py에서, glfwSetDropCallback을 통해 drop 행위에 대한 callback 함수인 drop_callback을 지정합니다. drop_callback에서는 총 3가지 기능을 수행합니다: ⓐ 애니메이팅 모드였을 경우 single mesh rendering 모드로 돌아옴 ⓑ drop된 파일의 경로를 바탕으로 obj 파일을 파싱해서 g_mesh의 클래스 멤버 변수를 업데이트 ⓒ 파싱된 데이터를 바탕으로 single mesh를 위한 VAO를 준비.

2. load_obj.py의 Mesh 클래스는 하나의 Mesh를 나타냅니다. Mesh 클래스는 다음과 같은 기능을 가집니다.
* parse obj file: load_obj.py:50-105에서, obj 파일의 첫 옵션이 v일때는 vertex position(및 vertex color)를 파싱해서 tmp_vertex_(pos/color - 없으면 white) 데이터에 저장합니다. vn일때는 vertex normal을, f일때는 face 정보를 파싱합니다. face를 파싱할 때, 만약 한 face에 3개 초과의 vertex가 사용되더라도, 어차피 한 face의 첫번째 vertex를 기준으로 조합가능한 삼각형을 모든 경우에 대해 그리면(line75에서 for문을 돌리듯) 해당 평면을 그릴 수 있습니다(예: 사각형을 삼각형 4개가 아닌 2개로만 그림). 그러므로 모든 경우에 대해 for 문을 돌려서 파싱하지 않고, 맨 첫 vertex를 기준으로 한번만 for문을 돌려서 반복적인 파싱 과정을 생략합니다. v, vn, f 외의 태그는 명세대로 무시합니다.

  load_obj.py:101-110에서, 추후 VBO에 사용될 vertex position, vertex color, vertex normal을 파싱된 face index 기준으로 정리해서 저장합니다. (EBO를 사용하기 위해 average normal vector를 계산하는 방법도 구현했었으나, obj 파일에서 vn이 normalize된 상태로 주어지므로 average를 구하는 과정에서 손실되는 데이터가 있는 것으로 판단되어, VBO 만으로 그리는 것으로 수정하였음)
  모든 파싱 및 저장이 완료되면, obj file 명, 전체 face, vertex 3~4개로 그리는 face, 그 외 face의 개수를 연산하여 콘솔 창에 출력해줍니다.

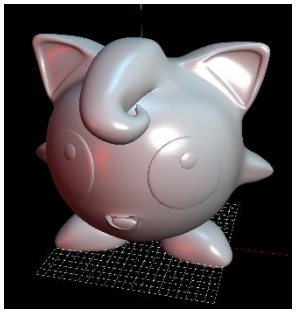
그림 2 purin.obj를 solid mode + light가 적용된 상태로 그린 것(기본은 wireframe mode)

* prepare VAO: load_obj.py:prepare_vao_mesh에서, 해당 mesh의 vertices 멤버 변수를 바탕으로 VBO에 vertex position, color, normal 벡터 정보를 vertex attribute array(location 0~2)에 저장해줍니다.

animating mode가 아니고 파일 파싱이 잘되어 VAO에 저장된 경우, window에 single mesh를 그려줍니다.

Implements: Animating hierarchical model rendering mode

model_loader.py의 ModelLoader 클래스는 계층 모델 애니메이션을 보여줍니다. 움직이는 피카츄를 중심으로 포켓몬볼을 각각 2개씩 가진 diglett과 zubat이 피카츄 주위를 맴돕니다. (https://youtu.be/1iuK2oKlVII) 각 obj 파일은 animating-models 폴더 안에 있으며, os 모듈을 사용하여 불러옵니다.

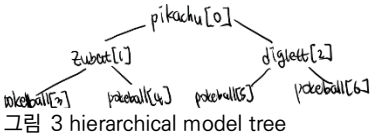

그림 3 hierarchical model tree

맨 처음 H 버튼을 눌러서 animating mode로 접근하면, main에서 선언된 ModelLoader 클래스 인스턴스인 animator가 계층 모델을 준비하기 위해 Node 클래스 인스턴스들을 생성합니다(model_loader.py:prepare_animating). 전체 노드 트리는 그림 3과 같습니다. Node 클래스는 node.py에 정의되어 있습니다. 각 node는 부모를 가리키는 변수 parent와 scale 값을 가집니다. 그리고, transform과 global_transform을 가지는데요, transform은 부모를 기준으로 변환된 것을 나타내며, global_transform은 계층 구조 전체에서 recursive하게 연산된 (root 기준으로 연산된) 것을 나타내는 행렬입니다. 추상적인 노드 구조와, 해당되는 obj 파일을 파싱한 mesh를 준비합니다.

pikachu	원 모양을 그리면서 천천히 translate
zubat	pikachu의 자식1. pikachu로부터 일정 거리 떨어져서(translate) 회전함(rotate)
diglett	pikachu의 자식2. pikachu로부터 일정 거리 떨어진 상태로 time에 의존하여 x축 방향으로 순환(translate)
pokeball(zubat)	zubat의 자식1. zubat으로부터 일정 거리 떨어져서(translate) 회전함(rotate)
pokeball(zubat)	zubat의 자식1. zubat으로부터 일정 거리 떨어져서(translate) 회전함(rotate)
pokeball(diglett)	diglett의 자식1. diglett으로부터 일정 거리 떨어져서(translate) 회전함(rotate)
pokeball(diglett)	diglett의 자식2. diglett으로부터 일정 거리 떨어져서(translate) 회전함(rotate)

이후에는, 계층 구조 모델을 그립니다. 매 프레임마다 main.py에서 animator.draw_hierarchical()을 호출하는데, 이때 각 노드의 부모를 기준으로 한 transformation이 model_loader.py:82-88에 정의되어 있고, 이후 root node에서 update_tree_global_transform을 통해 전체 노드의 global transformation을 부모 기준으로 left multiplication하여 업데이트 해줍니다. 그런 뒤, 각 노드에 해당하는 mesh를 그려줍니다. 이때, M(transformation) matrix에 global transform과 scale된 노드의 특성을 곱해서 연산함으로써 light에 의한 mesh face의 색상이 mesh의 transformation에 따라 변하도록(즉, light source가 world frame에서 고정된 것처럼 보이며, mesh가 회전한다고 해서 light에 의한 색이 object local frame에서 일정하지 않도록) 합니다. 추가적으로 M은 lighting할때 surface가 변환되는 것을 연산하고, normal vector가 변환되는 것 또한 연산하기 위해 uniform 변수인 M을 업데이트 하는데에도 쓰입니다. H를 다시 누르거나(이 경우 직전에 그린 single mesh를 그린 상태를 유지함) obj 파일을 드래그 앤 드랍하면 single mesh rendering 모드로 돌아가게 됩니다.

Implements: Light & Etc

1. 총 3개의 point light source를 가집니다. [ⓐ xyz(6,6,6) 위치의 흰색 rgba(1,1,1,1) light ⓑ xyz(0,20,0) 위치의 하늘색 rgba(0.52,0.81,0.92,1) light ⓒ xyz(-16,2,20) 위치의 빨간색 rgba(1,0,0,1) light] 수업에서는 특정 position에 광원이 위치하지만 directional light처럼 거리에 따라 빛의 세기가 감쇠되지 않는 light을 사용했는데, 좀더 자연스러운 point light을 사용하고 싶어서 빛이 거리에 따라 감쇠되는 연산을 추가하였습니다(main.py:100-105). 감쇠되는 상수는 임의로 설정하였습니다. 또한, world frame을 기준으로 light source가 고정된 것처럼 보이도록 하기 위해, uniform matrix M을 animating 모드에서는 계속해서 node의 transformation 매트릭스를 곱해서 업데이트 해줍니다. phong illumination model을 기준으로 ambient, diffuse, specular를 계산합니다(main.py:87-96). 또한, interpolation을 통해 픽셀 단위로 shading을 해줍니다(phong shading의, normal vector에 근거하여 연산하며 픽셀 단위의 작업을 위해 fragment shader 코드에서 연산함).

2. z 키를 눌러서 single mesh 렌더링 모드와 애니메이션 모델 두 모드에서 wireframe mode(polygon mode를 GL_LINE으로 설정)과 solid mode(polygon mode를 GL_FILL로 설정)를 토글할 수 있습니다.

Wireframe mode (light source 3개)	Solid mode (light source 3개)