R E P O R T

컴퓨터 그래픽스 팀 프로젝트 보고서

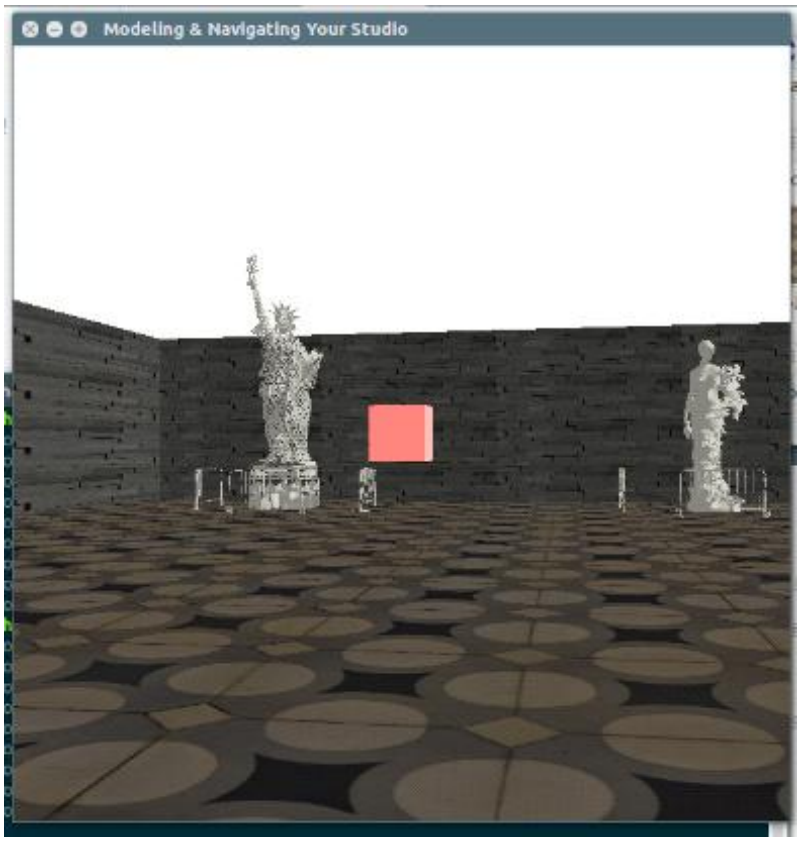
소프트웨어학부

20143043 김윤성
20143104 조승현

1. 조원 및 역할

20143043 김윤성 : 물체의 회전,확대,축소 변환, 텍스처 맵핑 구현
20143104 조승현 : 충돌 감지, FPS 게임적 요소 구현, 물체의 이동 구현

2. 박물관 모델링 결과



<실행 시 스크린 샷>

3. 프로젝트 구현 내용

1) 물체의 변환 : 컴퓨터 내부적으로 물체의 변환을 모두 행렬을 이용해 연산하기 때문에 물체들을 변환시키기 위해서는 행렬의 형식으로 표현해야 한다. 물체의 좌표를 월드 좌표계로 변환하기 위해서는 물체를 지켜보는 카메라의 내부 파라미터와 외부 파라미터에 해당하는 Projection 매트릭스와 View 매트릭스가 필요한데 모두 과제 파일 내부에 이미 구현되어 있는 Perspective, lookAt 함수를 이용하여 초기화 해주었다. 또 Model 매트릭스 에는 물체의 translation, rotation, scale 에 대해 연산하여 물체를 어디로 배치시키고 얼마나 회전을 할 지 어느 만큼 확대/축소 할지 정의 하였다. 이렇게 구성된 View 매트릭스와 Model 매트릭스를 곱한 것이 Model View 매트릭스가 되는데 이것은 물체가 카메라가 봤을 때 어떻게 표현되는가를 나타낸다, 여기에 Projection 매트릭스를 곱해주면 카메라가 보는 물체를 clipping space안에서 표현되도록 한다. Projection 매트릭스와 View 매트릭스는 다음과 같이 정의하여 사용하고 Model 매트릭스의 경우는 물체마다 다르게 배치, 회전 시켜야 하므로 각 물체마다 다르게 정의해준다.

```
// camera extrinsic param
mat_View = kmuvcl::math::lookAt(
    g_camera.position()(0), g_camera.position()(1), g_camera.position()(2),            // eye
    g_camera.center_position()(0), g_camera.center_position()(1), g_camera.center_position()(2)
    g_camera.up_direction()(0), g_camera.up_direction()(1), g_camera.up_direction()(2)
    );
// camera intrinsic param
mat_Proj = kmuvcl::math::perspective(g_camera.fovy(), 1.0f, 0.001f, 10000.0f);
```

2) 물체의 표현 : 물체는 빛과 재질의 상호작용에 의해 나오는 색상과 텍스처를 통해서 렌더링 된다. 빛과 재질의 상호작용은 빛이 얼마나 물체에 흡수되는지 , 흩어지는지와 물체의 색, 표면의 재질에 의해 결정된다. 우리가 어떤 물체를 보고 색을 느끼는 것은 물체의 표면에서 반사된 빛을 눈이 감지하는 것이다. 어떤 물체가 어떤 색이라고 느끼는 것은 해당 물체의 표면이 다른 색을 모두 흡수하고 해당 색만을 반사하기 때문에 우리의 눈은 그 색을 인지하게 된다. 컴퓨터 그래픽에서 현실세계의 물리적 특성을 모두 반영 할 수 없지만 Local lighting model을 이용하여 비슷하게 흉내 낼 수 있다. Phong-Reflection 모델은 Ambient reflection, Diffuse reflection, Specular reflection을 통해 빛의 작용을 계산한다면 각 픽셀에서 보이는 물체의 표면 색상을 결정 할 수 있도록 해준다.
더 나아가 물체 표면의 디테일은 텍스처를 물체에 맵핑 시켜 표현 해 준다. 텍스처란 물체의 표면에 입힐 이미지 데이터인데 물체의 특정 정점을 텍스쳐의 특정 정점에 대응 시키는 방식으로 이미지를 입혀준다.
그림에서 동상들은 카파 값을 이용하여 물체를 표현했고 벽과 바닥은 텍스쳐 매핑을 통해서 표현했다.

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texid_floor);
glUniform1i(loc_u_texid, 0);
mat_Model = kmuvcl::math::translate(0.0f, 0.0f, 0.0f) *
kmuvcl::math::rotate(360.0f, 0.0f, 00.0f, 0.0f) *
kmuvcl::math::scale(00.0f, 0.02f, 00.0f);
g_floor.draw3(mat_Model, mat_Proj, mat_View_inv);
```

3) 물체의 이동(편의성 고려) : HW_02를 통해서 마우스를 통해 물체를 선택 후에 키보드 방향키를 통해 해당 물체가 이동할 수 있도록 구현 하였다.

```
case GLUT_KEY_LEFT:
  obj_mat_Model[obj_select] = obj_mat_Model[obj_select] *
  kmuvcl::math::translate(-0.1f, 0.0f, 0.0f);
  break;
```

<obj_mat_Model 배열>

4) 충돌 감지 : 물체마다 각자의 중심 좌표를 가지고 있고 중심점 마다의 거리를 계산하여 일정 거리 이하가 되면 카메라를 다시 원래 좌표로 되돌린다. 충돌감지를 쉽게 하기 위하여 구 2개의 충돌을 감지하는 방법을 썼다.

```
float distance =
sqrt(pow(g_camera.position_(0)-libert_center_position(0),2) +
   pow(g_camera.position_(1)-libert_center_position(1),2) +
   pow(g_camera.position_(2)-libert_center_position(2),2)) ;

if(distance < d)
  return 1;
```

<collision 함수>

```
case GLUT_KEY_LEFT:
  g_camera.move_left(0.1f);
  if(collision())
    g_camera.move_right(0.1f);
  break;
```

<카메라 이동 제한>

5) FPS 요소 : 총알 오브젝트를 생성하여 특정 키보드 키가 눌릴 때 카메라가 보는 방향으로 총을 발사한다. 총알이 물체와 충돌하면 총알이 없어지도록 구현하였고 총알을 맞은 동상들은 색이 빨간색이 되도록 구현하였다..

```
case 'Q': case 'q':
  g_bullet.push_back(Bullet(kmuvcl::math::vec4f(g_camera.front_dir_(0), g_camera.front_dir_(1), g_camera.front_dir_(2), 1.0f),
                            kmuvcl::math::vec3f(g_camera.position_(0), g_camera.position_(1), g_camera.position_(2)),
                            1.0f, kmuvcl::math::vec4f(0.0f, 0.0f, 0.0f, 1.0f)));
break;
```

<q 누를 때 총알 instance>

```
if(g_bullet[i].collision(libert_center_position, jesus_center_position,
                         alucy_center_position,statue_center_position))
  g_bullet.erase(g_bullet.begin() + i);
```

<충돌 시 총알 삭제>