소프트웨어융합학과 20191102101 신동민

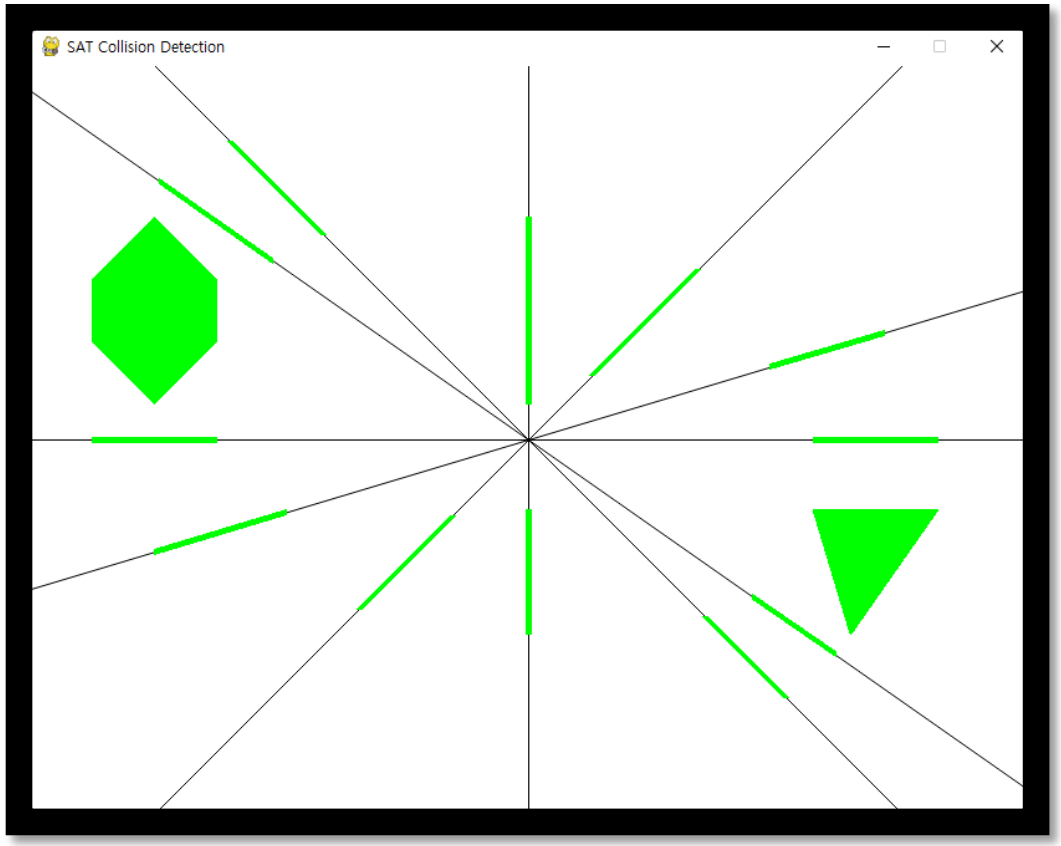
게임프로그래밍입문

게임엔진 제작 프로젝트

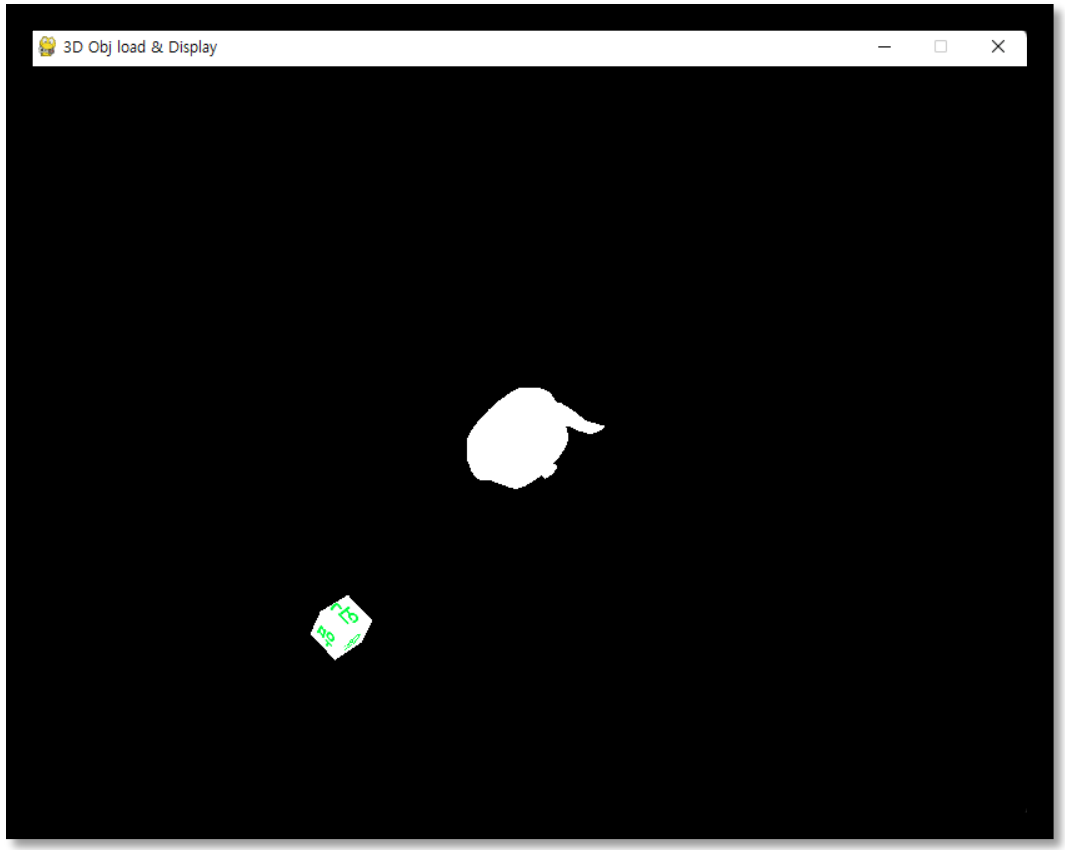
목차

제작한 프로젝트

1. 2D에서의 분리축 이론(Separate Axis Theorem)



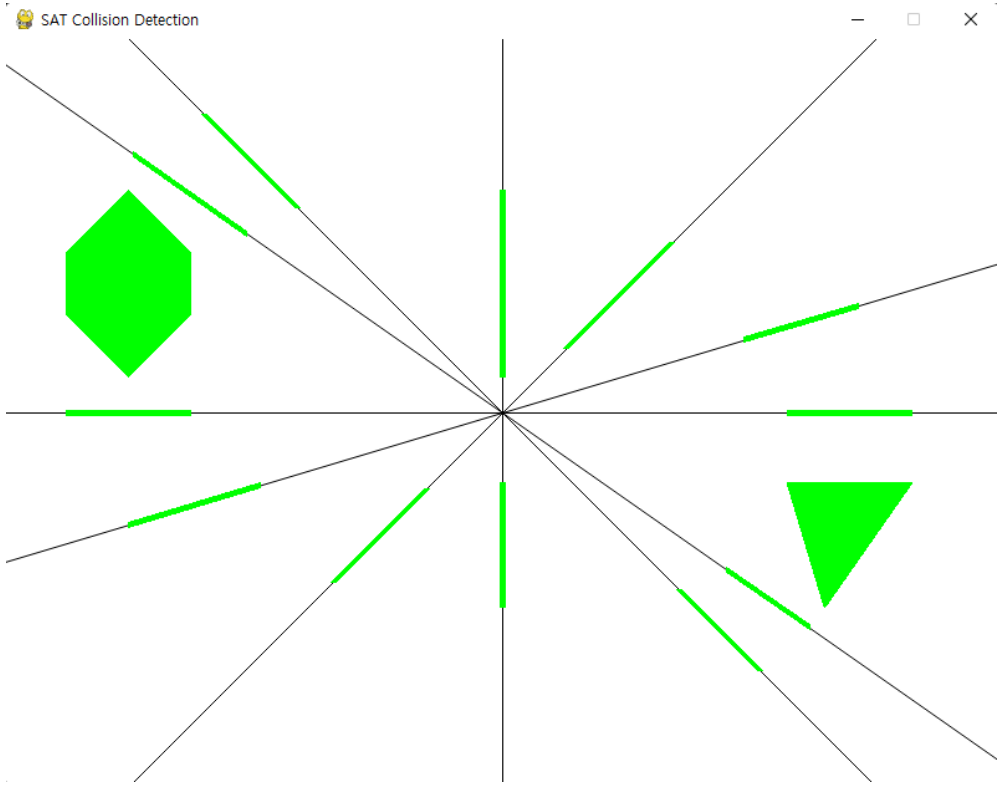
2. OBJ 파일 로드를 통해 두 개의 3D모델 Display



2D에서의 SAT

제작목표

- 2D에서의 Separate Axis Theorem을 구현하고, 각 축을 시각화해서 두 다각형이 충돌했을 때, 축에 대한 두 다각형의 내적이 어떻게 되는지 시각적으로 보여줄 수 있도록 함으로써, SAT에 대한 이해를 돕는다.



게임 엔진 디자인 & 구조

- 키보드의 WASD와 방향키로 이동시킬 수 있는 두 다각형을 보여주고, 두 다각형이 서로 충돌하면 빨간색으로 변한다.
- 화면의 중앙에 각 다각형의 분리축들을 보여준다. 분리축들은 두 다각형의 변들의 법선벡터이다.
- 두 다각형이 축에 대해 내적한 선도 보여준다. 도형이 움직임에 따라 선도 분리축을 따라 이동한다.
- py는 나누지 않고 한 파일에서 짰다.

코드 설명

```python
import pygame
import sys
import math

# Pygame 초기화
pygame.init()

# 화면 크기 및 색상 설정
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("SAT Collision Detection")
# 시간 설정
clock = pygame.time.Clock()
FPS = 60 # 시간 당 프레임
# 색상 정의
white = (255, 255, 255)
red = (255, 0, 0)
green = (0, 255, 0)
black = (0,0,0)

# 원점을 화면의 중앙으로 삼기 위한 세팅
originx = width / 2
originy = height / 2

# 다각형 정점 좌표
polygon1 = [[100, 100], [200, 100], [130, 200]]
polygon2 = [[-300, -150], [-350, -100], [-400, -150], [-400, -200], [-350, -250], [-300, -200]]
```

- 파이게임 초기 세팅을 한다. 화면의 크기는 800x600이고 60프레임이다.
- 추후에 다각형들에 대한 분리축을 볼 수 있도록 화면 중앙을 정점으로 여기는 originx, originy를 정해놓는다.
- 원점을 바꿔놓지 않고 분리축을 시각화하면 분리축들은 단위벡터로 정의되므로, 기존 파이게임의 원점인 화면의 좌측 상단 부분에서 나타나기 때문에 보이지 않는 분리축들이 발생하게 된다.
- 추가로 다각형의 정점도 정해놓는다. 폴리곤2의 정점에 마이너스가 들어가는 이유는 원점이 화면의 중앙을 원점으로 할 것이기 때문이다. 다각형의 정점은 convex 형태로 마음대로 쉽게 수정 가능하다.

코드 설명

```python
def project_polygon(axis, polygon):
    """다각형을 축에 사영한 결과를 반환합니다."""
    min_proj = float('inf')
    max_proj = float('-inf')

    for point in polygon:
        dot_product = axis[0] * point[0] + axis[1] * point[1]
        min_proj = min(min_proj, dot_product)
        max_proj = max(max_proj, dot_product)

    return min_proj, max_proj

```

- axis는 다각형들의 변들의 법선벡터이다.
- min_proj는 다각형의 정점들이 축에 대해서 내적했을 때 값이 가장 작을 때이므로, 초기값을 양의 무한대로 보내놓는다.
- max_proj는 다각형의 정점들이 축에 대해서 내적했을 때 값이 가장 클 때이므로, 초기값을 음의 무한대로 보내놓는다.

```python
def get_axes(polygon):
    """다각형의 변에 대한 노말축을 반환합니다."""
    axes = []
    for i in range(len(polygon)):
        point1 = polygon[i]
        point2 = polygon[(i + 1) % len(polygon)]
        edge = [point1[0] - point2[0], point1[1] - point2[1]]
        normal = [edge[1], -edge[0]]
        length = math.sqrt(normal[0]**2 + normal[1]**2)
        axis = [normal[0] / length, normal[1] / length]
        axes.append(axis)

    return axes

```

- 다각형의 정점을 차례로 이어서(두 점을 빼면 두 점사이의 벡터인 선이 구해진다.) 변들을 얻고, 변들을 통해서 법선 벡터들을 얻는다.
- 그 후 법선 벡터들을 정규화해서 반환한다. 이것이 충돌을 판정하기 위한 최소한의 분리축들이 된다.

코드 설명

```python
def check_collision(polygon1, polygon2):
    """두 다각형이 충돌하는지 여부를 확인합니다."""
    axes = get_axes(polygon1) + get_axes(polygon2)

    for axis in axes:
        min_proj1, max_proj1 = project_polygon(axis, polygon1)
        min_proj2, max_proj2 = project_polygon(axis, polygon2)

        if max_proj1 < min_proj2 or max_proj2 < min_proj1:
            # 축 사이에 갭이 있으면 충돌하지 않음
            return False

    return True

```

- SAT의 알고리즘이다.

- 두 다각형의 분리축들을 얻고 각각의 분리축들에 대해 두 다각형의 내적을 구해서 둘 사이가 겹치는 지 확인한다. 모든 축에 대해서 두 다각형의 내적들이 겹친다면, 두 다각형은 충돌한 것이다. (convex한 도형들만 가능하다.)

```python
def draw_SAT(polygon1, polygon2):
    """두 다각형의 각 선의 직교 벡터를 화면 중앙을 원점으로 시각화"""
    axes = get_axes(polygon1) + get_axes(polygon2)
    for axis in axes:
        x1 = (axis[0]* 1000 + originx)
        y1 = (axis[1]* 1000 + originy)
        x2 = (axis[0]* -1000 + originx)
        y2 = (axis[1]* -1000 + originy)
        pygame.draw.aaline(screen, black, [x1, y1], [x2, y2])
        """두 다각형이 축에 내적된 결과인 선을 시각화"""
        min_proj1 , max_proj1 = project_polygon(axis, polygon1)
        proj1_x1 = (axis[0]* min_proj1 + originx)
        proj1_y1 = (axis[1]* min_proj1 + originy)
        proj1_x2 = (axis[0]* max_proj1 + originx)
        proj1_y2 = (axis[1]* max_proj1 + originy)

        min_proj2 , max_proj2 = project_polygon(axis, polygon2)
        proj2_x1 = (axis[0]* min_proj2 + originx)
        proj2_y1 = (axis[1]* min_proj2 + originy)
        proj2_x2 = (axis[0]* max_proj2 + originx)
        proj2_y2 = (axis[1]* max_proj2 + originy)
        # 선끼리의 충돌 판정
        if max_proj1 < min_proj2 or max_proj2 < min_proj1:
            # 내적된 선이 서로 충돌하지 않으면 초록색
            pygame.draw.line(screen, green, [proj1_x1, proj1_y1], [proj1_x2, proj1_y2],5)
            pygame.draw.line(screen, green, [proj2_x1, proj2_y1], [proj2_x2, proj2_y2],5)
        else:
            # 충돌하면 빨간색
            pygame.draw.line(screen, red, [proj1_x1, proj1_y1], [proj1_x2, proj1_y2],5)
            pygame.draw.line(screen, red, [proj2_x1, proj2_y1], [proj2_x2, proj2_y2],5)
```

- 분리축들을 시각화하기 위한 함수다.
  우선 두 도형에 대한 분리축들을 얻는다.
- Originx와 originy를 더하지 않고 시각화하면 파이게임의 원점인 (0,0)에서 그려지므로, 벡터(1,0)과 같이 화면 바깥에 있는 분리축들이 보이지 않는다.
- 따라서 화면중앙에서 분리축들이 보일 수 있도록 originx와 originy를 더한다. 1000과 -1000을 곱한 이유는 분리축을 화면의 끝에서 끝까지 보일 수 있는 직선으로 그리기 위함이다.
- 분리축에 내적된 두 점도 화면의 중앙에 있는 분리축의 위치에 맞게 보일 수 있도록 originx와 originy를 더한다.
- 분리축에 내적한 값인 스칼라를 분리축에 곱해서 점을 구하고, 두 점을 이어서 선으로 그린다.

코드 설명

```
99
100 def draw_neworigin_polygon(polygon, color = green):
101     newpolygon = []
102     for point in polygon:
103         newpolygon.append([point[0] + originx, point[1] + originy])
104     pygame.draw.polygon(screen, color, newpolygon)
105
106
```

- 다각형의 실제 위치에 비해서 시각화된 분리축에 대한 내적 선은 원점을 옮긴 만큼의 위치에서 드러난다.
- 따라서 실제 위치에서 원점을 옮긴 만큼을 더한 자리에서 다각형을 그려야 분리축에 대한 내적 선과 알맞은 위치가 될 수 있다.

```
107 # 게임 루프
108 while True:
109     clock.tick(FPS)  # 프레임 제한
110     for event in pygame.event.get():
111         if event.type == pygame.QUIT:
112             pygame.quit()
113             sys.exit()
114
115     keys = pygame.key.get_pressed()
116     if keys[pygame.K_LEFT]:
117         for i in range(len(polygon1)):
118             polygon1[i][0] -= 1
119     if keys[pygame.K_RIGHT]:
120         for i in range(len(polygon1)):
121             polygon1[i][0] += 1
122     if keys[pygame.K_UP]:
123         for i in range(len(polygon1)):
124             polygon1[i][1] -= 1
125     if keys[pygame.K_DOWN]:
126         for i in range(len(polygon1)):
127             polygon1[i][1] += 1
128
129     keys = pygame.key.get_pressed()
130     if keys[pygame.K_a]:
131         for i in range(len(polygon2)):
132             polygon2[i][0] -= 1
133     if keys[pygame.K_d]:
134         for i in range(len(polygon2)):
135             polygon2[i][0] += 1
136     if keys[pygame.K_w]:
137         for i in range(len(polygon2)):
138             polygon2[i][1] -= 1
139     if keys[pygame.K_s]:
140         for i in range(len(polygon2)):
141             polygon2[i][1] += 1
142     # 충돌 감지
143     collision = check_collision(polygon1, polygon2)
```

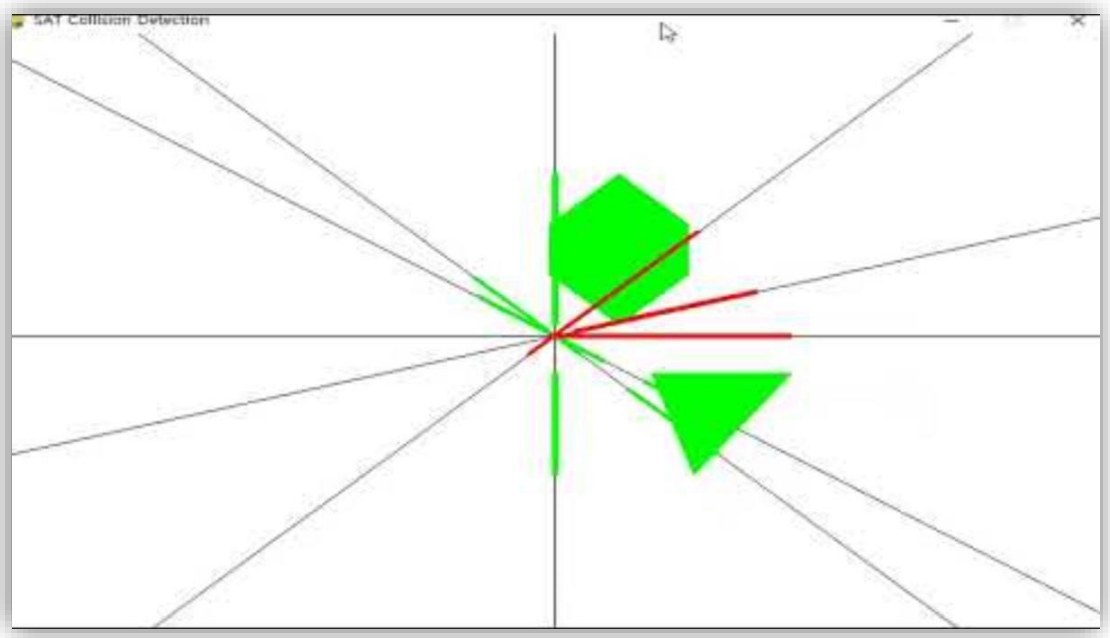
- 종료버튼을 누르면 프로그램이 꺼진다.
- WASD를 누르면 두번째 다각형이 움직인다. (두번째 다각형은 왼쪽에 위치해 있다.)
- 방향키를 누르면 첫번째 다각형이 움직인다. (첫번째 다각형은 오른쪽에 위치해 있다.)
- 두 다각형의 충돌을 감지한다.

코드 설명

```
144
145     # 화면 그리기
146     screen.fill(white)
147     # pygame.draw.polygon(screen, green, polygon1)
148     # pygame.draw.polygon(screen, green, polygon2)
149     draw_neworigin_polygon(polygon1)
150     draw_neworigin_polygon(polygon2)
151     draw_SAT(polygon1, polygon2)
152     # 충돌 시 색상 변경
153     if collision:
154         draw_neworigin_polygon(polygon1, red)
155         draw_neworigin_polygon(polygon2, red)
156
157     pygame.display.flip()
158
159 # 코드 실행
160 pygame.quit()
161
```

- 배경은 흰색으로 하고, 다각형과 SAT를 그린다. 다각형은 기본으로 녹색이고 충돌 시 빨간색이 된다.
- SAT에 내적된 선 또한 기본은 녹색이고, 선끼리의 충돌 시 빨간색이 된다.

실행 영상



실행 영상

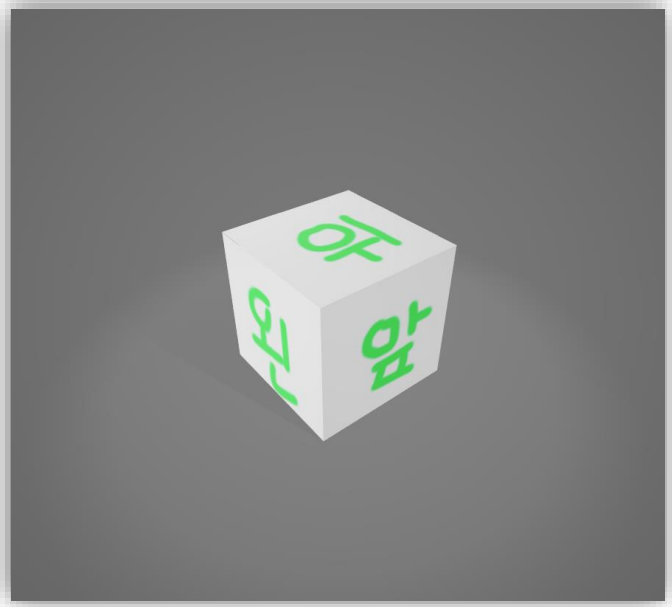
OBJ 파일 로드를 통해 두 개의 3D모델 Display

제작목표

- .obj 파일을 읽어서 PyOpenGL에서 Display한다.
- Texture가 있는 큐브를 Load하고, 복잡한 정점들을 가진 주전자를 Load해서 원하는 대로 Transform한다.

게임 엔진 디자인 & 구조

- 주전자는 원점에서 일정 방향으로 회전한다.
- 텍스처를 가진 큐브는 위치 변환된 자리에서부터 자전하면서 원점을 중심으로 공전한다.
- obj파일 두 개와 텍스처로 쓰일 png파일
- mtl파일은 사용하지 않았다.
- py는 나누지 않고 한 파일에서 짰다.



코드 설명

```
# Blender 3.3.1
# www.blender.org
mtllib model.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
vn -0.0000 1.0000 -0.0000
vn -0.0000 -0.0000 1.0000
vn -1.0000 -0.0000 -0.0000
vn -0.0000 -1.0000 -0.0000
vn 1.0000 -0.0000 -0.0000
vn -0.0000 -0.0000 -1.0000
vt 0.625000 0.500000
vt 0.375000 0.500000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.875000 0.500000
vt 0.625000 0.250000
vt 0.125000 0.500000
vt 0.375000 0.250000
vt 0.875000 0.750000
vt 0.625000 1.000000
vt 0.625000 0.000000
vt 0.375000 0.000000
vt 0.375000 1.000000
vt 0.125000 0.750000
s 0
usemtl Material
f 5/5/1 3/3/1 1/1/1
f 3/3/2 8/13/2 4/4/2
f 7/11/3 6/8/3 8/12/3
f 2/2/4 8/14/4 6/7/4
f 1/1/5 4/4/5 2/2/5
f 5/6/6 2/2/6 6/8/6
f 5/5/1 7/9/1 3/3/1
f 3/3/2 7/10/2 8/13/2
f 7/11/3 5/6/3 6/8/3
f 2/2/4 4/4/4 8/14/4
f 1/1/5 3/3/5 4/4/5
f 5/6/6 1/1/6 2/2/6
```

```python
import pygame
import sys
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *


# OBJ 파일을 읽어서 정점, 면, 텍스처 좌표, 법선 벡터 정보를 추출하는 함수
def load_obj(file_path):
    vertices = []
    faces = []
    texture_coords = []
    normals = []
    with open(file_path, 'r') as file:
        for line in file: # 파일로부터 한 줄씩 얻어온다.
            if line.startswith('v '): # obj를 보면 v_ (_ = 띄어쓰기) 이후에 값들이 들어가 있다. 따라서 인덱스로 2부터 얻는다.
                vertices.append(list(map(float, line[2:].split()))) # split()은 문자열을 띄어쓰기로 구분하여 리스트로 만든다.
                #우리가 원하는 float값으로 바꿔주기 위해서 map을 사용하고, 다시 list로 변환해준다.
            elif line.startswith('vt '):
                texture_coords.append(list(map(float, line[3:].split()))) # vt_ 이므로 3부터
            elif line.startswith('vn '):
                normals.append(list(map(float, line[3:].split())))  # vn_이므로 3부터
            elif line.startswith('f '): # F는 v/vt/vn 의 인덱스들로 이루어져 있다.
                face_info = line[2:].split() # v/vt/vn 씩 얻기 (현재 읽으려는 model.obj는 v/vt/vn가 3개씩 들어있으므로 3개씩 들어감.)
                face = []
                texture_coord = []
                normal = []
                for vertex_info in face_info:
                    vertex_data = vertex_info.split('/') # v/vt/vn 씩 얻었으므로 /을 기준으로 나눔
                    face.append(int(vertex_data[0])) # v부터
                    if len(vertex_data) >= 2 and vertex_data[1]:
                        texture_coord.append(int(vertex_data[1]))
                    if len(vertex_data) >= 3 and vertex_data[2]:
                        normal.append(int(vertex_data[2]))
                faces.append([face, texture_coord, normal])
    return vertices, faces, texture_coords, normals
```

- 모델을 로드할 때 파일 경로로 파일을 읽는다.
- Obj파일의 v, vt(UV), vn, f 정보는 비슷한 패턴을 가진다.

코드 설명

```python
# 텍스처를 로드하는 함수
def load_texture(texture_path):
    texture_surface = pygame.image.load(texture_path)
    texture_data = pygame.image.tostring(texture_surface, 'RGBA', 1) # 이미지를 바이트 버퍼로

    texture_id = glGenTextures(1) #  텍스처 이름을 생성.(사용할 텍스처는 하나이므로 1)
    glBindTexture(GL_TEXTURE_2D, texture_id) # 사용하려는 텍스처를 GL_TEXTURE_2D에 바인딩. = (이후에 GL_TEXTURE_2D에 세팅되는 것들이 texture_id에 적용됨.)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, texture_surface.get_width(), texture_surface.get_height(), 0,
                 GL_RGBA, GL_UNSIGNED_BYTE, texture_data)
    # 2차원 텍스처 이미지를 지정하는 함수. 각 매개변수에 알맞도록.

    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR) # 텍스처 축소 함수는 텍스처 처리 중인 픽셀이 텍스처 요소보다 큰 영역에 매핑될 때마다 사용된다.
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR) # 텍스처화되는 픽셀이 하나의 텍스처 요소보다 작거나 같은 영역에 매핑될 때 사용된다.

    return texture_id
```

```python
# OBJ 파일의 정점, 면, 텍스처 좌표, 법선 벡터 정보를 사용하여 모델을 그리는 함수
def draw_textured_obj(vertices, faces, texture_coords, normals, texture_id):
    glEnable(GL_TEXTURE_2D) # 텍스처를 적용할 수 있도록 GL_TEXTURE_2D 활성화
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL) # 텍스처 환경 매개 변수 설정
    glBindTexture(GL_TEXTURE_2D, texture_id) # GL_TEXTURE_2D에 texture_id 바인딩

    glBegin(GL_TRIANGLES)
    for face in faces:
        for i, vertex_index in enumerate(face[0]): # enumerate는 튜플인 (인덱스, 리스트의 원소) 형태로 반환한다.
            vertex = vertices[vertex_index - 1]
            texture_coord = texture_coords[face[1][i] - 1]
            normal = normals[face[2][i] - 1]
            # 현재 텍스처가 활성화 되어있으므로 glTexCoord2fv로 현재 텍스처 좌표를 설정한다.
            glTexCoord2fv(texture_coord)
            glNormal3fv(normal)
            glVertex3fv(vertex)
    glEnd()
    glDisable(GL_TEXTURE_2D)
```

- 텍스처를 로드하고, 텍스처가 입혀진 모델을 그릴 때 해당 텍스처 모드를 활성화해서 적용한다.

코드 설명

```
70
71  # OBJ 파일의 정점과 면 정보를 사용하여 모델을 그리는 함수
72  def draw_obj(vertices, faces):
73      glBegin(GL_TRIANGLES)
74      for face in faces:
75          for vertex_index in face[0]: # 한 face엔 3개의 vertex_index가 있으므로, GL_TRIANGLES에 맞게 3번씩 glVertex3fv함.
76              vertex = vertices[vertex_index - 1]
77              glVertex3fv(vertex)
78      glEnd()
```

- 텍스처를 사용하지 않는 모델을 그리는 함수도 만들어둔다.

```
80  # Pygame 초기화 및 화면 설정
81  pygame.init()
82  display = (800, 600)
83  pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
84  pygame.display.set_caption("3D Obj load & Display")
85  # 카메라 및 뷰포트 설정
86  gluPerspective(45, (display[0] / display[1]), 0.1, 200.0)
87  glTranslatef(0.0, 0.0, -50)
```

- Pygame 초기 세팅을 하고, 카메라와 뷰포트를 설정한다.
- 모델이 좀 크므로 z 위치를 -50로 보낸다.

```
89  # OBJ 파일 및 텍스처 파일 경로
90  first_obj_file_path = 'model.obj'
91  texture_file_path = 'face.png'
92
93  second_obj_file_path = 'teapot.obj'
94  # OBJ 파일 로드
95  first_vertices, first_faces, first_texture_coords, first_normals = load_obj(first_obj_file_path)
96
97  second_vertices, second_faces, _, _ = load_obj(second_obj_file_path)
98  # 텍스처 로드
99  texture_id = load_texture(texture_file_path)
100
101
102 # 초기 회전 각도 설정
103 rotation_angle1 = 0
104 rotation_angle2 = 0
```

- Obj파일과 텍스처 파일의 경로를 설정해서 파일을 로드한다.
- First는 텍스처를 사용하므로 vt가 필요하다. (큐브 모양의 텍스처가 입혀진 모델이다.)
- second는 텍스처를 사용하지 않으므로 vertex와 face만 로드한다. (주전자 모양의 모델이다.)
- 메인 루프에서 쓰일 회전을 위해서 초기 회전 각도를 설정해둔다.

코드 설명

```
106  # 메인 루프
107  while True:
108
109      for event in pygame.event.get():
110          if event.type == pygame.QUIT:
111              pygame.quit()
112              sys.exit()
113
114      glEnable(GL_DEPTH_TEST)  # 깊이 테스트 활성화
115      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
116      # 여러 개의 모델에 원하는 transform을 위한 push&pop
117      glPushMatrix()
118      glRotatef(rotation_angle1, -1, 1, 0)  # 첫 번째 모델 회전
119      glTranslatef(10.0, 10.0, 0.0)  # 첫 번째 모델의 위치 조정
120      glRotatef(rotation_angle2, 3, 1, 1)  # 첫 번째 모델 회전
121      draw_textured_obj(first_vertices, first_faces, first_texture_coords, first_normals, texture_id)
122      glPopMatrix()
123
124
125      glPushMatrix()
126      glRotatef(rotation_angle2, 3, 1, 1)  # 두 번째 모델 회전
127      glScalef(0.3,0.3,0.3)
128      draw_obj(second_vertices, second_faces)
129      glPopMatrix()
130
131      # 회전 각도 갱신
132      rotation_angle1 += 1
133      rotation_angle2 += 5
134
135      pygame.display.flip()
136      pygame.time.wait(10)
```

- 창닫기를 누르면 프로그램이 꺼지도록 한다.
- 깊이 테스트를 활성화하지 않으면 텍스처가 입혀진 모델의 face가 이상하게 보이므로, 활성화해준다.
- 여러 개의 모델을 사용하므로, pushmatrix와 popmatrix를 사용한다.
- 첫 번째 모델은 원점에 있을 때, (3,1,1)축을 기준으로 5도씩 회전하므로 자전한다. 그 후, (10,10,0)의 위치에 있도록 한다. 그 후, (-1,1,0)축을 기준으로 회전하므로 공전하게 된다.
- 두 번째 모델은 원점에서 (3,1,1) 축을 기준으로 5도씩 회전한다.
- 두 번째 모델은 크기가 너무 크므로, 스케일을 0.3씩 줄인다.

실행 영상

실행영상

감사합니다