CG HW3 Report

P76111131 唐飴苹

Scene Graph

- Environment

- OS : Ubuntu 22.04 / Linux
- Python 3.9.15
- Libraries :
  - pywavefront
  - pygame 2.1.2

- Method Desdcription

load cube.obj

load cube.obj by using `pygame.Wavefront()` ,and then draw cubes by the information of vertices loaded in `scene`

```
scene = pywavefront.Wavefront('cube.obj', collect_faces=True)

def DrawCube(scale, trans, rotate):
    glPushMatrix()
    glTranslatef(trans[0]*2, trans[1]*2, trans[2]*2)
    glRotatef(rotate[0], rotate[1], rotate[2], rotate[3])
    glScalef(scale[0], scale[1], scale[2])

    for i, mesh in enumerate(scene.mesh_list):
        glBegin(GL_POLYGON)
        for face in mesh.faces:
            for vertex_i in face:
                glVertex3f(*scene.vertices[vertex_i])
        glEnd()
    glPopMatrix()

def DrawCubebyScale(scale):
    glPushMatrix()
    glScalef(scale[0], scale[1], scale[2])

    for i, mesh in enumerate(scene.mesh_list):
        glBegin(GL_POLYGON)
        for face in mesh.faces:
            for vertex_i in face:
                glVertex3f(*scene.vertices[vertex_i])
        glEnd()
    glPopMatrix()
```

Model Scene graph

- model - root
  - body
    - mouse
    - tail
    - hair
    - front right leg 1
      - front right leg 2
    - front left leg 1
      - front left leg 2
    - back right leg 1
      - back right leg 2
    - back left leg 1
      - back left leg 2

Build the model

build the scene graph of model according to the example function below :

```
def root_of_body():
  glPushMatrix()

  glPushMatrix()
  glTranslate(...)
  glRotatef(...)
  DrawCube()
  glPopMatrix()

  # child node of body ...

  glPopMatrix()
```

- How to run the program?

```
$ python main.py
```

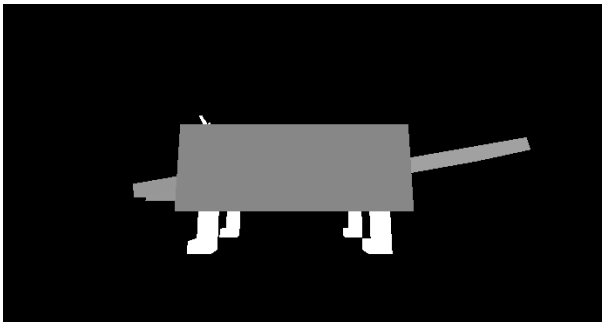
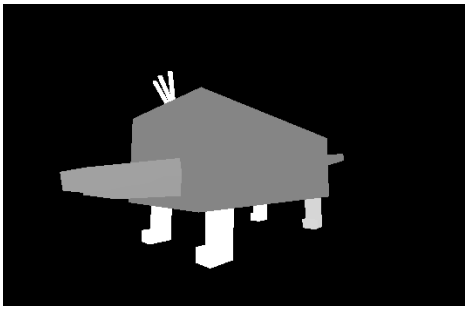
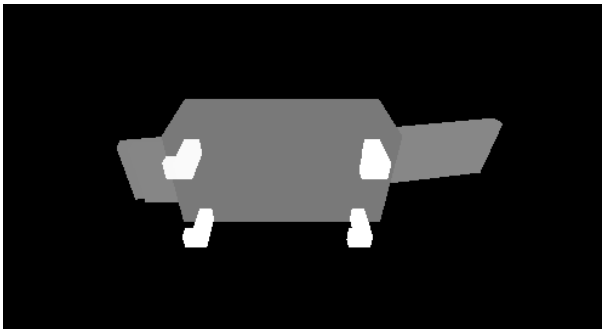
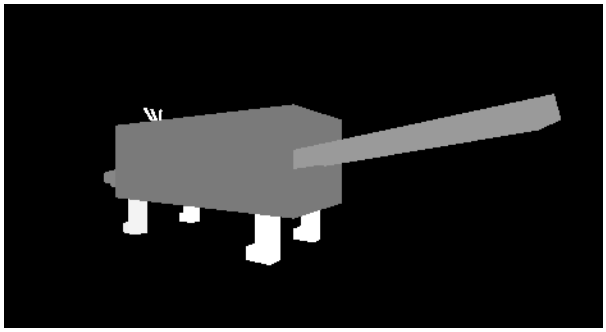
Usage

- Camera
  - **scrolling the mouse wheel** to zoom the field of view
- Model controling
  - **press left mouse button and move** to change perspectives

    - **click ENTER** to perform running action in the original place, **click it again** to stop
    - **click SPACE** to perform running action forward, **click it again** to stop
- Joint of Model Moving
    - **click Q** to rotate up the front_right_leg1, **click it again** to stop
    - **click W** to rotate down the front_right_leg1, **click it again** to stop
    - **click A** to rotate up the front_right_leg2, **click it again** to stop
    - **click S** to rotate up the front_right_leg2, **click it again** to stop

- Result

- model

Skeletal Animation

- Environment

- OS : Ubuntu 22.04 / Linux
- g++ 9.4.0
- OpenGL (4.6.0)
- Libraries

- glfw (3.3.8)
- glm (0.9.9.8)
- assimp (5.0.1)

- Method Desdcription

use assimp to load `model.dae` given by TA

model.h

- struct `Vertex`
  - recording `postition`, `normal`, `texCoords`
- struct BoneMatrix
  - recording `offset`, `final_transform`
- struct VertexBones
  - recording `ids`(indices influced the vertex), `weights`

- class `Mesh`
  - recording `vertices`, `indices`, `vao`, `vbo`, … mesh information
  - function `draw()` : bind vertex array and draw elements
- class `Model`
  - read `model.dae` and get the information from the file
    - including `bone_location`, `bone_matrices`, `inverse_transform` …
  - function `readNodeHierarchy()` : set final transform of each bone

camera.h

set camera view controling by scrolling and move the cursor

Shader.h

read and compile the vertex and fragment shader program files

- **vertex shader program** : model.vs

  use uniform variables to caculate the position of joints

```
#version 330 core

uniform mat4 model;
uniform mat4 projection;
uniform mat4 view;
const int MAX_BONES = 240;
uniform mat4 bones[MAX_BONES];
```

```
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texcoord;
layout (location = 3) in ivec4 bone_ids;
layout (location = 4) in vec4 weights;

void main(void){
  mat4 bone_transform = bones[bone_ids[0]] * weights[0];
  bone_transform += bones[bone_ids[1]] * weights[1];
  bone_transform += bones[bone_ids[2]] * weights[2];
  bone_transform += bones[bone_ids[3]] * weights[3];

  vec4 pos = (projection * view * model) * bone_transform * vec4(position, 1.0);
  gl_Position = pos;
}
```

- **fragment shader program** : model.frag

  set the color of model

```
#version 330 core

layout(location=0) out vec4 color;

void main(){
    color = vec4(1.0f, 1.0f, 0.5f, 0.0f);
}
```

- How to run the program?

```
# compile
$ g++ main.cpp -o program -lGLEW -lglfw -lGL -lX11 -lpthread -lXrandr
-lXi -ldl -lassimp

# execute
$ ./program
```

Usage

- Camera
  - **scrolling the mouse wheel** to zoom the field of view
- Model controling
  - **press left mouse button and move** to change perspectives
  - **press SPACE** to perform running action in the original place, **press it again** to stop
- Joint of Model Moving
  - head
    - **press 1** to rotate right ; **press 2** to rotate left
    - **press 3** to rotate up ; **press 4** to rotate down
  - chest

- **press 5** to rotate right ; **press 6** to rotate left
- **press 7** to rotate up ; **press 8** to rotate down

- front_left_leg
  - **press R** to rotate up the lower leg ; **press F** to rotate down the lower leg
  - **press T** to rotate up the upper leg ; **press G** to rotate down the upper leg
- front_right_leg
  - **press Y** to rotate up the upper leg ; **press H** to rotate down the upper leg
  - **press U** to rotate up the lower leg ; **press J** to rotate down the lower leg
- tail
  - **press Z** to rotate left ; **press X** to rotate right
  - **press C** to rotate up ; **press V** to rotate down
- wing
  - **press B** to rotate up the left wing; **press N** to rotate doent the left wing
  - **press M** to rotate down the right wing ; **press ,** to rotate up the right wing

- Result