COMP3069 UNNC Project

Student Name: *Jingyu Ma* *scyjm2@nottingham.edu.cn*

Student ID: *20031510*

Date of Submission: *December 12, 2020*

Word Count: *10 pages*

Individual Project Title: Factory with assembly line

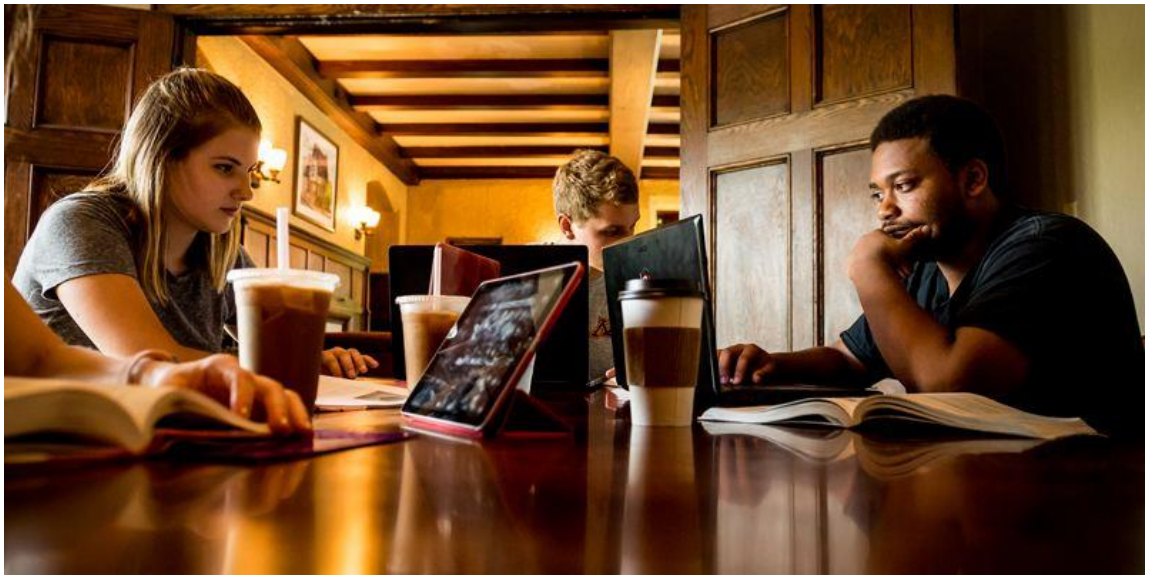
Description: *This is a demo about a scene in the factory about processing materials*

Declaration:

I confirm that this coursework submission is all my own work, except where explicitly indicated within the text.

1.Introduction

This is a demo inside a processing factory , and the camera was inspired by COVID-19 , every classroom has a camera , other models are for decoration and fulfillment of the scene , this should be an automation robot factory and no one should supervise it.
This project should include 3D models , transformation of objects , view-points , animations , texturing ,lighting and some creative ideas as well .



2.Structure

The structure is organized as follows :

- Several 3D models, which may contain some imported objects, and some own-created ones,
- Transformation of models (scaling, translating, rotating),
- Different viewpoint to the scene environment,
- Animations for some objects in the scene,
- Texturing –you need to employ texture to make some models look more realistic,
- Lighting – you should apply lighting effect in your scene,
- Sound system and creative ideas

3 .3D models

3.1 Introduction to Obj format model:

OBJ file is a standard 3D model file format developed by Alias|Wavefront for its workstation-based 3D modeling and animation software "Advanced Visualizer". At present, most mainstream software platforms on the market support the OBJ format. The OBJ model has therefore become a commonly used data exchange format. OBJ file is a kind of text file. Since the model is marked in plain code, it is convenient for us to view and edit it.

3.2 **Obj model features**

The OBJ3.0 file format supports straight lines (Line), polygons (Polygon), surfaces (Surface) and free-form curves (Free-form Curve). Straight lines and polygons are described by their points. Curves and surfaces are defined according to their control points and additional information attached to the curve type. This information supports regular and irregular curves, including those based on Bezier curves (Bezier) , B-spline, Cardinal/Catmull-Rom and Taylor equations. Other features are as follows:

(1) The OBJ file is a 3D model file. Does not contain animation, material properties, texture paths, dynamics, particles and other information.

(2) OBJ files mainly support Polygons models. Although Curves, Surfaces, and Point Group Materials are also supported, the OBJ file exported by Maya does not include this information.

(3) OBJ file supports more than three points, which is very useful. Many other model file formats only support three-point faces, so the model imported into Maya is often triangulated, which is very unfavorable for us to reprocess the model.

(4) OBJ file supports normal and texture coordinates. After adjusting the texture in other software, the texture coordinate information can be stored in the OBJ file, so that after the file is imported into Maya, you only need to specify the texture file path, and there is no need to adjust the texture coordinates.

3.3 Obj model download website

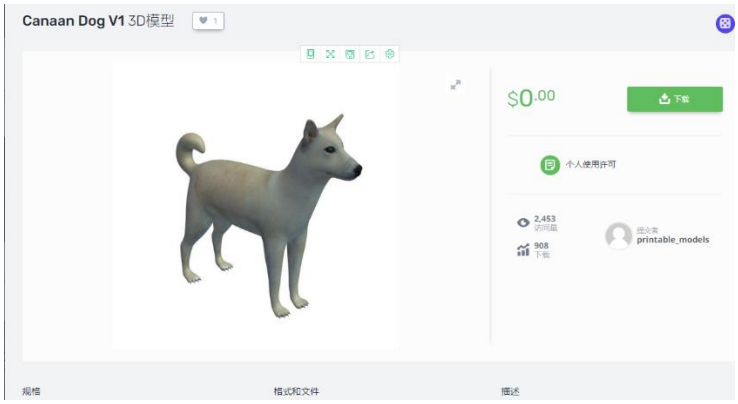
https://free3d.com/3d-model/

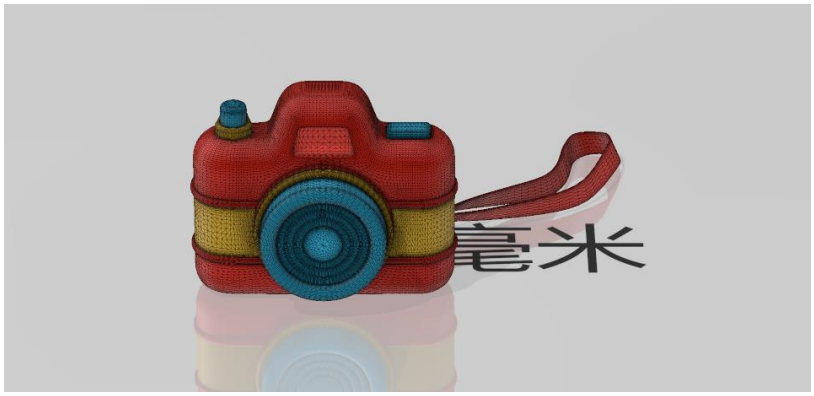
3.4 Obj model analysis library

If you write an OBJ model analysis library by yourself is a very complicated process, and it is not easy to adapt to the various structures of the OBJ model, so I use the open source library to

analyze the obj model, the specific path of the open source library:
https:// github.com/tinyobjloader/tinyobjloader
The use of this library is a very simple process, and teacher use cases have been given on GitHub, so I use this library as the obj model parser. I use std_image to read texture image information.

There are in total three models inside the scene , since the scene is quite small too much models will make the scene involved and complicated and not aesthetic . I put a camera at the center of the scene , a dog model and a robot model . There are listed as follows.





There are many ways to load obj into openGL , some resources online use a library called Assimp to load in obj files , After all models are loaded by Assimp, they are saved to a scene object. The scene object contains a root node that contains a number of child nodes, each of which contains grid data. The Scene object also contains grid data and material data. Similar

mechanism was used in our project but instead of this , I created a class called tiny_obj_loader.h , and feed the data into the datalist then use function glCallList() to show the object .Sample code are here as follows.

```
void MyOBJ::draw_obj()
{
    glCallList(datalist);
}
```

```
GLuint texid;
void MyOBJ::load_obj(const char *filename, const char *imgname)
{
    std::string err;
    std::string warn;
    if (!tinyobj::LoadObj(&attrib, &shapes, &materials, &warn, &err, filename, NULL, true, true)) {
        throw std::runtime_error(err);
    }

    //init texture
    int width, height, channel;
    stbi_set_flip_vertically_on_load(true);
    unsigned char *data = (unsigned char *)stbi_load(imgname, &width, &height, &channel, 0);
    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &mtexid);
    glBindTexture(GL_TEXTURE_2D, mtexid);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    gluBuild2DMipmaps(GL_TEXTURE_2D, channel, width, height, GL_RGB, GL_UNSIGNED_BYTE, data);
    stbi_image_free(data);

    datalist = glGenLists(1);
    glNewList(datalist, GL_COMPILE);
    glBindTexture(GL_TEXTURE_2D, mtexid);
    glBegin(GL_TRIANGLES);
    for (const auto& shape : shapes) {
        for (const auto& index : shape.mesh.indices) {
            int vidx = index.vertex_index;
            int tidx = index.texcoord_index;
            glTexCoord2f(attrib.texcoords[2 * tidx + 0],
                attrib.texcoords[2 * tidx + 1]);
            glVertex3f(attrib.vertices[3 * vidx + 0],
                attrib.vertices[3 * vidx + 1],
                attrib.vertices[3 * vidx + 2]);
        }
    }
    glEnd();
    glBindTexture(GL_TEXTURE_2D, 0);
    glEndList();
}
```

```
// draw obj
glColor3f(0.0, 0.0, 0.5); //change color
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(x_camera, y_camera, z_camera); // add there to move the camera
glRotatef(90.0f, 0, 0, 1);//3.1415926 / 2
glRotatef(90.0f, 0, 1, 0);//3.1415926 / 2
glScalef(0.2, 0.2, 0.2);
cam_obj.draw_obj();
```

Remember before you call such function , please allocate all the texture into one picture instead of several pictures and then use load obj to load in the model.

4. Transformation

You can control the camera using 1-2 number key , and the robot using 3-7 3,4,5,6 for front-back-up-down , and 7 for rotation . This is the transformation of model and you can see the sphere should be transformed around the assembly line , which you could toggle by the menu to start the animation of moving and process the material. And when the model is imported into the scene , you should use scaling , rotating , and transformation to do the work .

And when constructing the floor and walls you also have to use the transformation to manually place everything in place.
The transformation is carried out by the function glTranslatef(),glScalef(),glRotatef() and so on.

5.Viewpoint

In the adjustment of the angle of view, I call it the camera, because the function that can be achieved is roaming, so how is roaming realized? Below I will briefly describe the implementation of roaming. First, I call the gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0) function, which defines a view matrix, And multiply with the current matrix. The first group of eyex, eyey, eyez cameras are in world coordinates, the second group of centerx, centery, centerz is the position of the camera lens in world coordinates, and the third group of upx, upy, upz cameras are in world coordinates. You imagine the camera as your own head: the first set of data is the position of your head, the second set of data is the position of the object you see, and the third set is the direction the head is facing (because you can tilt your head and look at the same object)

```
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(-2.75, 4, 9.25);
glScalef(1, 8, 1);
glutSolidCube(0.5);

glColor3f(1.0, 1.0, 0.0);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(-2.75, 4, 5.25);
glScalef(1, 8, 1);
glutSolidCube(0.5);

glColor3f(1.0, 1.0, 0.0);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(0, 6.25, 9.25);
glScalef(12, 1, 1);
glutSolidCube(0.5);

glColor3f(1.0, 1.0, 0.0);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(0, 6.25, 5.25);
glScalef(12, 1, 1);
glutSolidCube(0.5);

glColor3f(1.0, 1.0, 0.0);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(2.75, 4, 9.25);
glScalef(1, 8, 1);
glutSolidCube(0.5);


glColor3f(1.0, 1.0, 0.0);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(2.75, 4, 5.25);
glScalef(1, 8, 1);
glutSolidCube(0.5);
```

6.Animation

In the design of animation, I mainly use timer to control. First, I call glutTimerFunc(200, &timerFunc, 12); this function, which starts every 200 milliseconds, and calls back the void timerFunc(int value) function. So how is the animation formed? First of all, I set the animation in the form of enabling. For example, I made a right-click menu, that is, when I choose to right-click to start the animation, I start to translate the x-axis direction of the processed object continuously Next, when I reach the switching direction, I set the direction of the y-axis, and then switch the coordinates of the x-axis, so that the timer is continuously added in a loop, and a simple animation is drawn.

7.Texturing

Texture design is a very interesting process. First of all, where is the place I need to design the texture, that is, the wall and the floor and my shield, then the first thing I need to do is to read the texture information stbi_set_flip_vertically_on_load(false); data = (unsigned char *)stbi_load("b4_arms.jpg", &width, &height, &channel, 0); Load the texture information, after reading the texture information I set the texture data

```
stbi_set_flip_vertically_on_load(false);
data = (unsigned char *)stbi_load("3.jpg", &width, &height, &channel, 0);
glGenTextures(1, &gtexid1);
glBindTexture(GL_TEXTURE_2D, gtexid1);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
gluBuild2DMipmaps(GL_TEXTURE_2D, channel, width, height, GL_RGB, GL_UNSIGNED_BYTE, data);
stbi_image_free(data);
```

This completes the reading of the texture data and the binding of the texture coordinates. After the data is completed, all I need to do is to bind the texture to a specific location

```
glColor3f(0.0, 0.0, 0.5);
glLoadIdentity();
gluLookAt(eyex, eyey, eyez, eyex + dirx, eyey + diry, eyez + dirz, 0, 1, 0);
glTranslatef(0, 0, -16);
glBindTexture(GL_TEXTURE_2D, texid);
glBegin(GL_QUADS);
glTexCoord2f(0.0f, 0.0f); glVertex2f(-10.5f, -16.0f);
glTexCoord2f(1.0f, 0.0f); glVertex2f(27.0f, -16.0f);
glTexCoord2f(1.0f, 1.0f); glVertex2f(27.0f, 20.0f);
glTexCoord2f(0.0f, 1.0f); glVertex2f(-10.5f, 20.0f);
glEnd();
glBindTexture(GL_TEXTURE_2D, 0);
```

8.Lighting

The lighting system is divided into three parts, namely light source, material and lighting environment.

The light source is the source of light, which can be the sun or electric light mentioned above.

Material refers to the surface of various objects that receive light. Since how an object reflects light is only determined by the surface of the object (light refraction is not considered in OpenGL), the characteristics of the material determine the characteristics of the object's reflected light.

The lighting environment refers to some additional parameters that will affect the final lighting picture. For example, after some light is reflected multiple times, it is no longer possible to distinguish which light source it is emitted from. At this time, specify an "ambient brightness" parameter to enable The resulting picture is closer to the real situation.

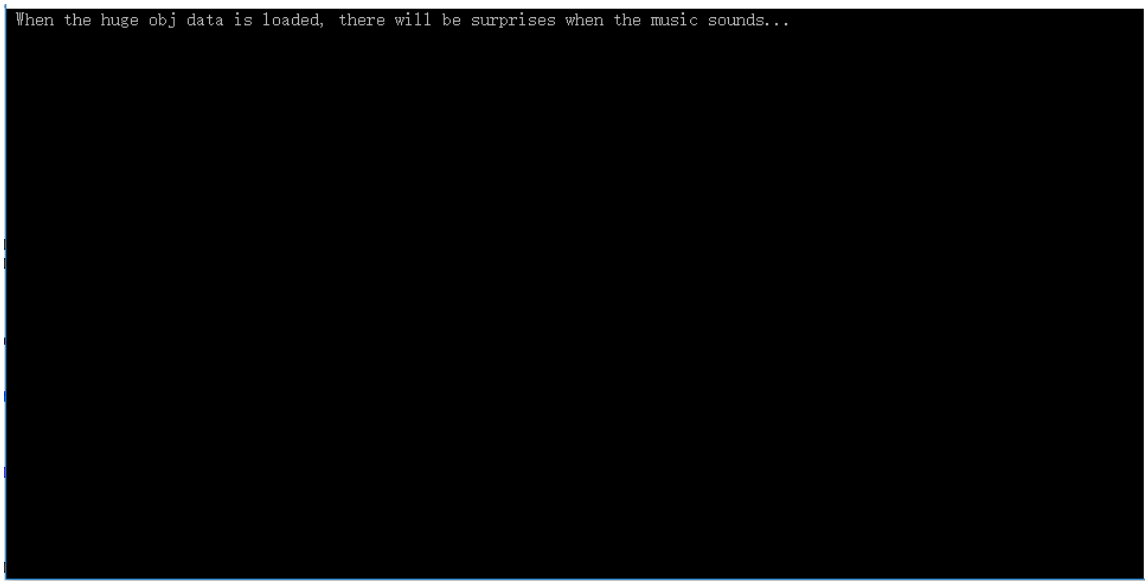
```
void openLightRes() {
    GLfloat lightPosition[] = { 0.0f, 0.0f, 1.0f, 0.0f };
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
}

void closeLightRes() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    GLfloat light_position[] = { 0.0, 0.0, 1.5, 1.0 };
    GLfloat light_ambient[] = { 0.0, 0.0, 1.0, 1.0 };
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    GLfloat mat_amb_diff0[] = { 0.0, 0.0, 1.0, 1.0 };
    GLfloat mat_amb_diff1[] = { 1.0, 0.0, 0.0, 1.0 };
    glDisable(GL_LIGHTING);
    glDisable(GL_LIGHT0);
}
```

9.Sound System

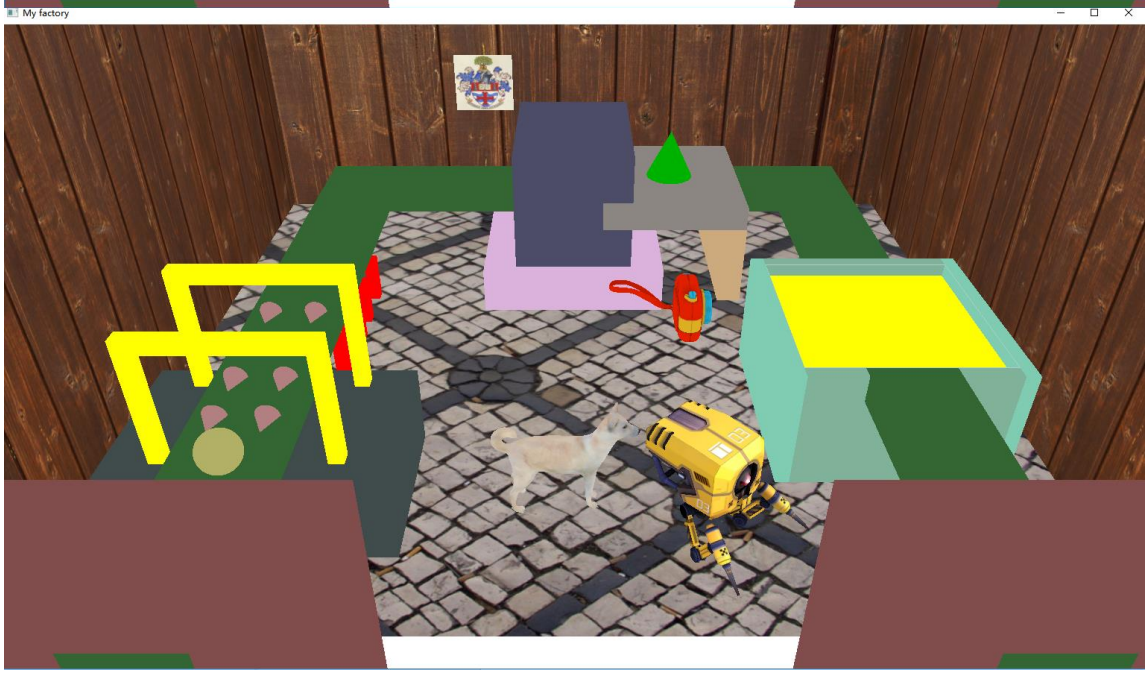
The sound system is a very simple process, because I just call the sound playback function in Windows, and this function just plays the audio data in wav format, and this sound mainly contains dog barking, because I set it in my model The model of the dog, the sound of the camera shooting because I used the camera model and the sound of the robot because my design involves robots, so the detailed code is used.

```
void mouse(int button, int state, int x, int y)
{

    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        PlaySound(L"dog.wav", NULL, SND_ASYNC);
    }
    else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
    {
        PlaySound(L"camera.wav", NULL, SND_ASYNC);
    }

    glutPostRedisplay();
}
```

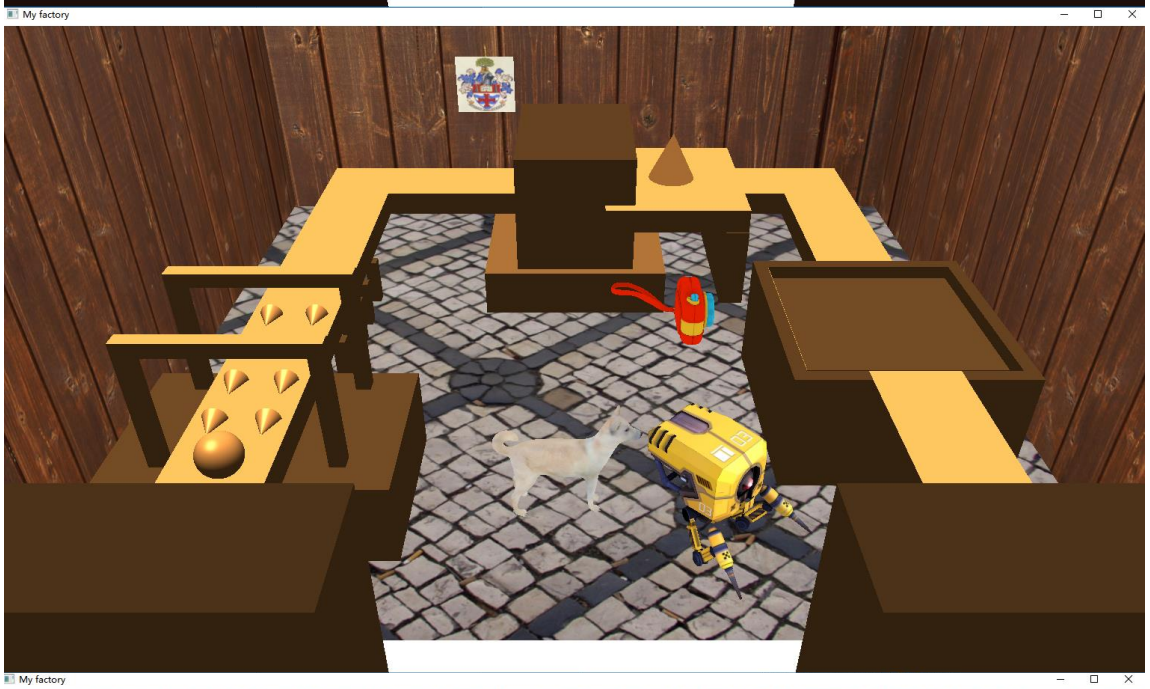
10. Results screenshot

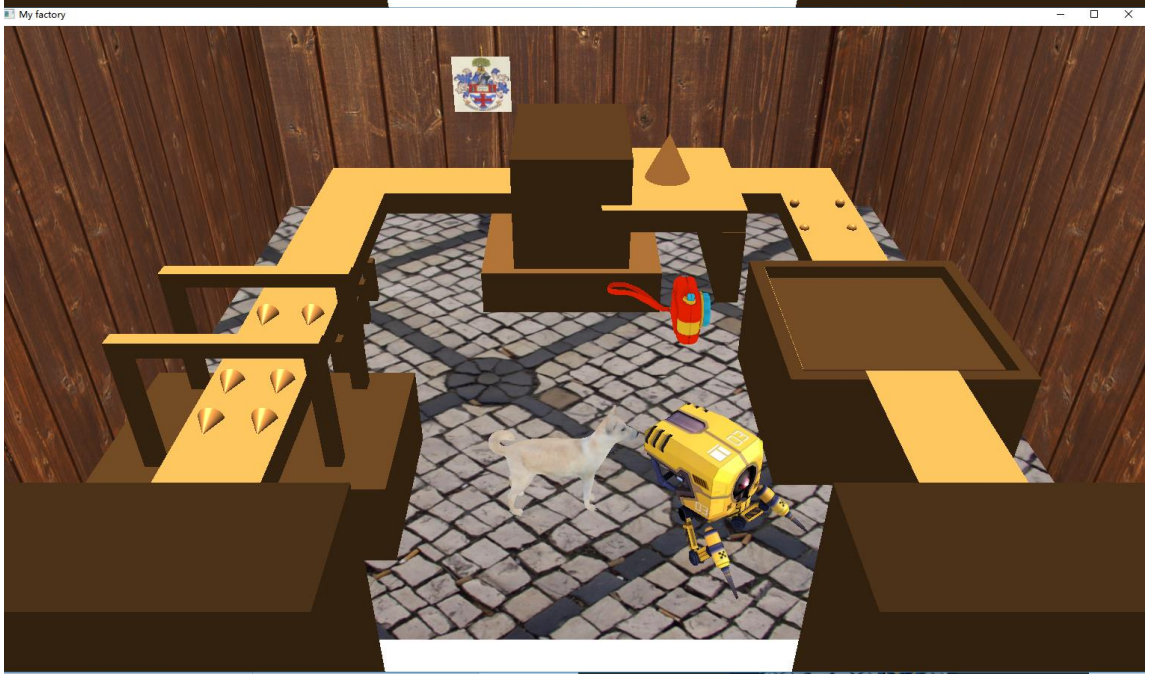

When the huge obj data is loaded, there will be surprises when the music sounds...


My factory


My factory


My factory


My factory


My factory


My factory


My factory