

# Room Escape Game

## Team 2

20131064 김동훈

20141440 이승원

20151506 정하빈

## 1. Introduction

We thought making a game is a good project to apply knowledge that we have learned through this semester. Thus, we made a simple room escape game with first person viewpoint. Various technique such as VBO, texture, and lighting are used in this game. To clear the game, a player should answer the correct numbers which are hidden in the objects. Since we made a special selection view mode when a player clicks the objects, the player can observe the objects in detail. Thus, the player has to explore the room and examine the objects.

## 2. Object & Password Control

## 2-1. Arrangement with Blender

We used Blender to handle objects. It helps us to resize, relocate or combine objects altogether. All objects we used for the project are free modeling objects that are released on the internet. We made the stage by putting the objects together properly using Blender.

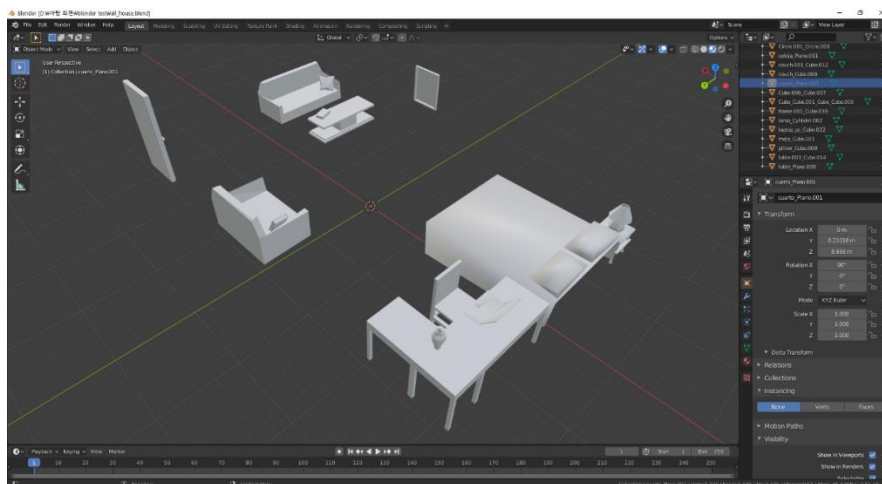


Figure 1) overall view of the stage with Blender

## 2-2. Object Loading & Rendering

Typical obj file has a following format. Letter comes first such as 'v' for vertex, 'vn' for vertex normal, 'vt' for vertex texture and 'f' for face index. Next, numbers are given. For example, in case of 'v', Three numbers are given, which represent x,y and z coordinate respectively. Thus, we separated each case by using strcmp() and then stored the values. Below shows the part of loadOBJ() function in our code.

```
v -2.011660 5.999999 6.400001
v 5.670000 4.970498 -3.407556
```

Figure 2) format of vertex information in object file

```
if (strcmp(lineHeader, "v") == 0) {
    vec3 vertex;
    int lineData = fscanf(file, "%f %f %f\n", &vertex.x, &vertex.y, &vertex.z);
```

Figure 3) code implementation to handle vertex information

After loading information to variables, we rendered the objects using VBO. We generated vertex array VAO first to create space to hold value. Then we bind the information, that we stored to variables above, to it. Finally, we rendered the objects by using its information. Since it exploits GPU not CPU, the overall performance of our programs increased.

```
/* generate VAO */
glGenVertexArrays(1, &object->vao);
glBindVertexArray(object->vao);

/* generate VBO */
glGenBuffers(1, &object->vbo);
glBindBuffer(GL_ARRAY_BUFFER, object->vbo);
glBufferData(GL_ARRAY_BUFFER, object->vertices.size() * sizeof(vec3), &object->vertices[0], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Figure 4) generate VAO and bind information to it

```
/* bind VAO & EBO */
glBindVertexArray(object->vao);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, object->ebo);
glDrawElements(GL_TRIANGLES, (GLsizei)object->indices.size(), GL_UNSIGNED_INT, (void*)0);
```

Figure 5) render objects

### 2-3. Print Text

We implemented the code in two cases. One case is for displaying tip text on the upper left of the screen, which is implemented by `glutBitmapString()` function. The other case is implemented by `glutStrokeString()` function. It is used when players insert the password to the keypad. It seems like players really insert numbers to the keypad, but, in fact, we positioned the text properly giving players illusion that he/she pushes the numbers of the keypad.

```
/* bitmap mode */
if (mode == 'b') {
    string s = text;
    glRasterPos2i(x, y);
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, str);
}

/* stroke mode */
else if (mode == 's') {
    glLineWidth(2.0);
    glTranslated(x, y, 0);
    glScaled(0.2, 0.2, 0);
    glutStrokeString(GLUT_STROKE_ROMAN, str);
}
```

Figure 6) code implementation for displaying text

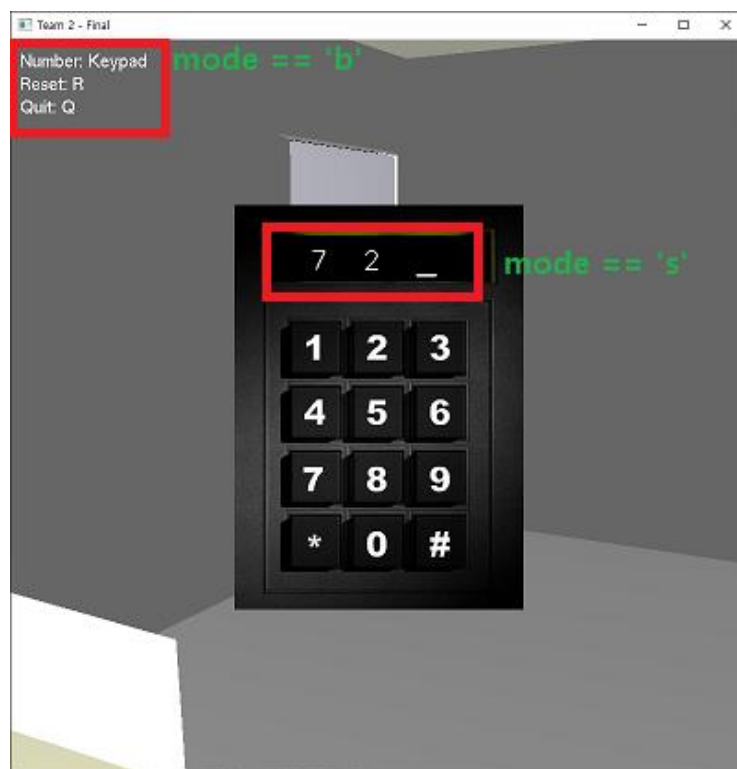


Figure 7) two cases for displaying text

## 2-4. Password Implementation

When a player pushes a number, corresponding digit is stored to global variable 'pwChar'. In addition, no matter what key a player presses, 'saveCount' becomes 1 (true). If 'saveCount' is true, digit in 'pwChar' is transformed from int to character by itoa() function and appended to 'pwBefore'. 'pwBefore' continuously append a digit that player presses. If three digits inserted but it is not a correct answer, 'pwBefore' is initialized to empty string. This algorithm repeats until a player answers the correct numbers.

```
else if (key == '1') { pwChar = 1; pwCount--; saveCount = 1; }
else if (key == '2') { pwChar = 2; pwCount--; saveCount = 1; }
else if (key == '3') { pwChar = 3; pwCount--; saveCount = 1; }

/* password incorrect */
else if (pwCount >= 0) {
    pwString = "    Biggest...?";

    if (pwCount < 3) {
        if (saveCount) {
            pwBefore = pwBefore + " " + itoa(pwChar, temp, 10);
            saveCount = 0;
        }

        pwString = insertPassword(pwCount);
    }

    printText(x, y, vec3(1.0, 1.0, 1.0), 's', pwString.c_str());
}

/* password reset */
else {
    Sleep(300);
    pwCount = 3;
    pwBefore = "";
}
```

< key press >

< password algorithm >

Figure 8) password overall algorithm with keyboard

### 3. Viewport & Trackball

#### 3-1. Movable First-person Camera View

As we implement a room escape game, we had to mimic human perspective. The position of camera must be movable by pressing keyboard buttons (WSAD), and the view direction of camera must be changeable by moving mouse. So, we needed to implement both and combine them into one camera system.

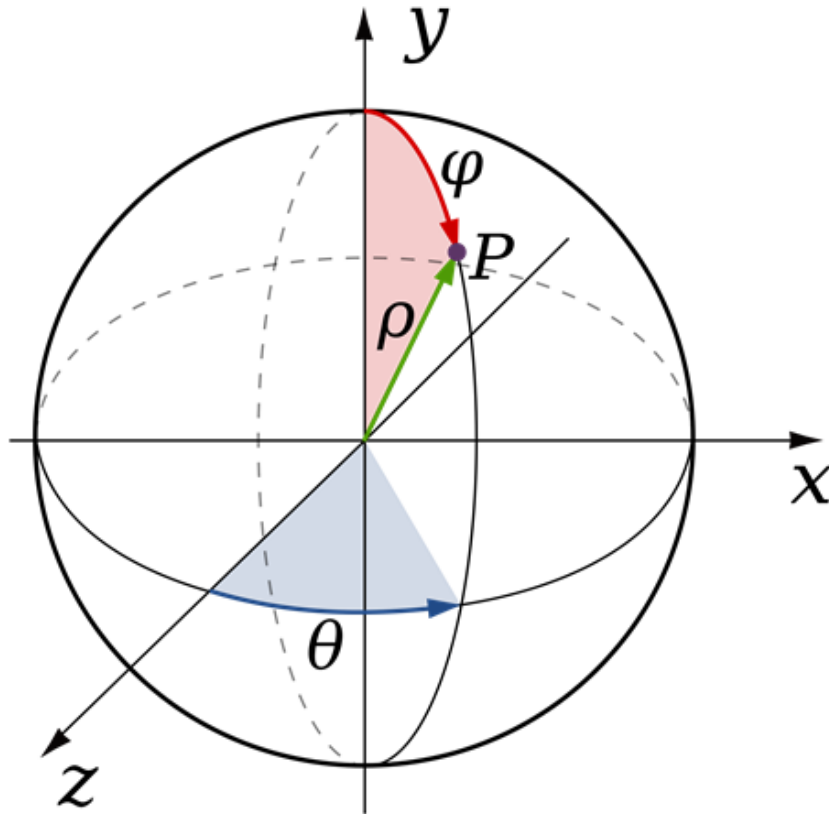


Figure 9) OpenGL 3D coordination figure

First, every point on a unit sphere which have its radius 1, can be represented as a tuple of two angles, theta and pi. So, we can define our view direction as a vector which starts from the origin O, and ends at a specific point P. From now, let's name that vector as OP vector. Let pi be the angle between y axis and OP vector and let theta be the angle between z axis and the projection of OP vector on xz plane.

Then, we can easily find out the xyz coordination of the point P by rotating the vector (0,0,1) about x axis by  $(\pi/2 - \pi)$ , and then by rotating the result about y axis by theta. Now, we can define our view direction, OP vector.

```

void passiveMouseMove(int x, int y)
{
    float mouseMove_x = (float)(mousePos[0] - x);
    float mouseMove_y = (float)(mousePos[1] - y);
    float rotateRate = 2400.0f;
    float tempAt[3];

    glutSetCursor(mouseMode);
    glutWarpPointer(screenSize / 2, screenSize / 2);
    mousePos[0] = screenSize / 2;
    mousePos[1] = screenSize / 2;

    theta += (float)(6.28318 * mouseMove_x / rotateRate);
    if (theta >= 6.28318) theta -= 6.28318f;
    else if (theta <= -6.28318) theta += 6.28318f;

    pi += (float)(1.570795 * mouseMove_y / rotateRate);
    if (pi >= 1.570795) pi = 1.570795f;
    else if (pi <= -1.570795) pi = -1.570795f;

    rightVec = normalize(cross(at, up));

    at.x = 0;
    at.y = sin(pi);
    at.z = -cos(pi);

    tempAt[0] = cos(theta) * at.x + sin(theta) * at.z;
    tempAt[1] = at[1];
    tempAt[2] = -sin(theta) * at.x + cos(theta) * at.z;

    at.x = tempAt[0];
    at.y = tempAt[1];
    at.z = tempAt[2];
}

```

Figure 10) Actual OpenGL C++ codes

As you from at the first four blocks, we maintain theta and pi as global variables, and mouse movement along x and y coordination are added to theta and pi respectively after some calculation. And also you can see from the last 3 blocks, we rotated the vector (0,0,1) about x axis by pi and then about y axis by theta. Plus, we used glutPassiveMotionFunc for the function to track mouse movement without clicking any buttons.

Next, because our view direction is defined, we need to get right vector from that.

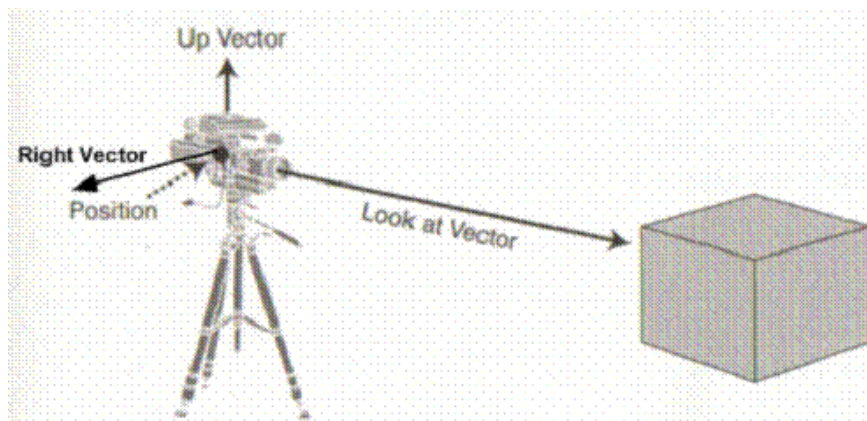


Figure 12) OpenGL camera view coordination

Cross-producting our view direction (look at vector in the figure) with vector (0,1,0) is fine because we are mimicking human perspective. Also because of our mimicking, we must not change y coordination of the camera position. So we need to make y coordination of both vectors and normalize them. Now, by adding or subtracting the view direction vector or right vector with zero y coordination to or from the camera position, we can define our new camera position.

```
rightVec = normalize(cross(at, up));
```

```
void keyboard(unsigned char key, int x, int y)
{
    float moveRate = 2.0;
    vec3 temp_at = normalize(vec3(at.x, 0, at.z));
    vec3 temp_right = normalize(vec3(rightVec.x, 0, rightVec.z));

    if (key == 'w' || key == 'W') {
        eyePosition += temp_at / moveRate;
    }
    else if (key == 's' || key == 'S') {
        eyePosition -= temp_at / moveRate;
    }
    else if (key == 'a' || key == 'A') {
        eyePosition -= temp_right / moveRate;
    }
    else if (key == 'd' || key == 'D') {
        eyePosition += temp_right / moveRate;
    }
    glutPostRedisplay();
}
```

Figure 12) Actual OpenGL C++ codes

As you can see here, we removed y coordination of view direction vector and right vector, then renormalized them to maintain y coordination of the camera position (eyePosition vector in the picture). Now we can add the new view direction vector (temp\_at vector in the picture) to the camera position to go toward the view direction, or new right vector (temp\_right vector in the picture) to go right. We can also subtract them from the camera position to go backward or left.

### 3-2. Selection view (fake trackball)

Because we wanted to make possible for players investigate the objects to find some hints, we needed to implement trackball to show every part of the objects. So, we decided to use fake trackball. Because we used pre-placed .obj files, we need to translate them back to the origin then rotate and scale them and then translate them back to the original position.

```
float rotateRate = 10.0f;  
rot.x -= (float)(mouseMove_y / rotateRate);  
rot.y -= (float)(mouseMove_x / rotateRate);
```

Figure 13) Maintaining rotation angle

```
void mouseWheel(int wheel, int direction, int x, int y)  
{  
    if (direction > 0)  
        rot.z += 0.03f;  
  
    else if (direction < 0) {  
        rot.z -= 0.03f;  
        if (rot.z < 0.5f) rot.z = 0.5f;  
    }  
  
    glutPostRedisplay();  
}
```

Figure 14) Mouse wheel callback function for maintaining scale factor

```
void selectionView(vec3 pos, vec3 offset)  
{  
    glTranslatef(pos.x, pos.y, pos.z);  
    glRotated(rot.y, 0.0f, 1.0f, 0.0f);  
    glRotatef(rot.x, 1.0f, 0.0f, 0.0f);  
    glScalef(rot.z, rot.z, rot.z);  
    glTranslatef(offset.x, offset.y, offset.z);  
}
```

Figure 15) Function for defining matrix

```
selectionView(targetPosition, vec3(-21.27, -3.74, -8.45));  
glColor3f(0.8, 0.7, 1.0);  
drawOBJ(obj_book);
```

Figure 16) Actual code for drawing object



## 4. Selection

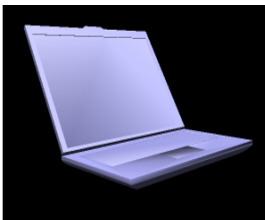
Object selection is the distinguish method which object model are within some specific volume of space. The main idea is that save the matrix of picking area to select buffer and draw selected object with each object name. And then, render the selected object after calculating z-depth.

### 4-1. Object Name

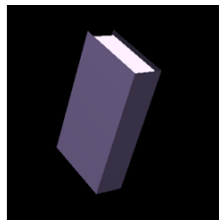
```
drawAllObjects()
{
    glPushMatrix();
    glPushName(obj_book->ID);
    {
        glColor3f(0.8, 0.7, 1.0);
        drawOBJ(obj_book);
    }
    glPopName();
    glPopMatrix();
}
```

Figure 17) draw each object code

Before draw each object, all objects should have each object name by `glPushName(GLuint name)`. Because OpenGL could distinguish what object is clicked vertex. By the `constructOBJ(string fname, obj_t* object, GLuint ID)`, we already defined the name of each object with object initialization. So, we can call each object name by `struct obj_t->ID` and give the name by `glPushName(obj_t->ID)`. Here are some objects name examples.



object : laptop  
name : obj\_laptop->ID



object : book  
name : obj\_book->ID



object : lamp  
name : obj\_lamp->ID

Figure 18) object naming examples

## 4-2. Pick Matrix with Click

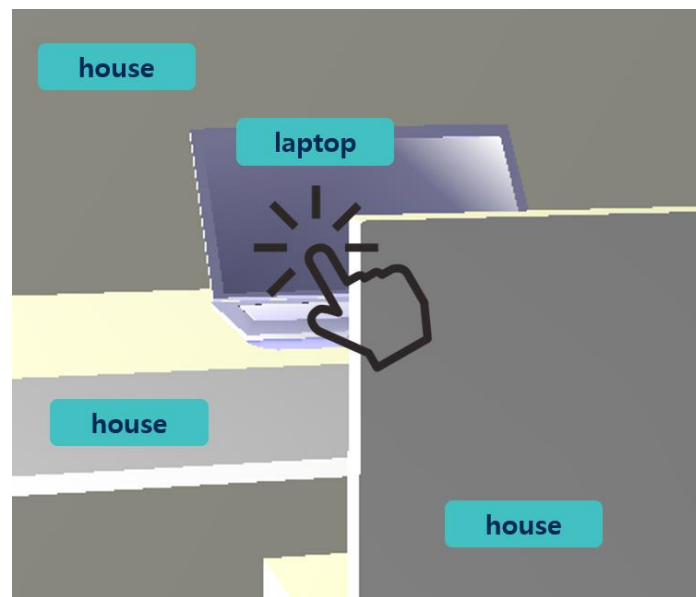


Figure 18) example of mouse clicking

If mouse click is occurred, then mouse makes z-axis ray and shoots to front z-axis direction until arrived at last 3D object model. At above example, the mouse clicks laptop of center. But OpenGL can get all object from starting z-axis point of mouse click. So, mouse click ray will get the objects as laptop and house that beyond of laptop. At this example, we just defined desk and chair as house by blender. It is the reason that we don't want to classify desk and chair for selecting.

```
pickObject()
{
    glSelectBuffer(bufferSize, selectBuf);
    (void)glRenderMode(GL_SELECT);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    {
        glLoadIdentity();
        gluPickMatrix((GLdouble)x, (GLdouble)(viewport[3] - y),
                     pickSize, pickSize, viewport);
        gluPerspective(40.0, shapeRatio, 1.0, 300.0);
        drawAllObjects(GL_SELECT);
    }
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();

    hits = glRenderMode(GL_RENDER);
    processHits(hits, selectBuf);

    glutPostRedisplay();
}
```

Figure 19) overall of pickObject() algorithm

After mouse click, the `pickObject()` function gets glut mouse activation. And OpenGL will get matrix information of pick area by changing the render mode from `GL_RENDER` to `GL_SELECT`. About `gluPickMatrix()`, `x` and `y` are position of mouse clicking but the starting point of `y` is left-bottom, so calculate to real height with `viewport[height] - y`. Then, we can get the click position. After this, mouse generate square which width is `pickSize` and height is `pickSize`. The image as below helps for easy understanding of `gluPickMatrix()`.

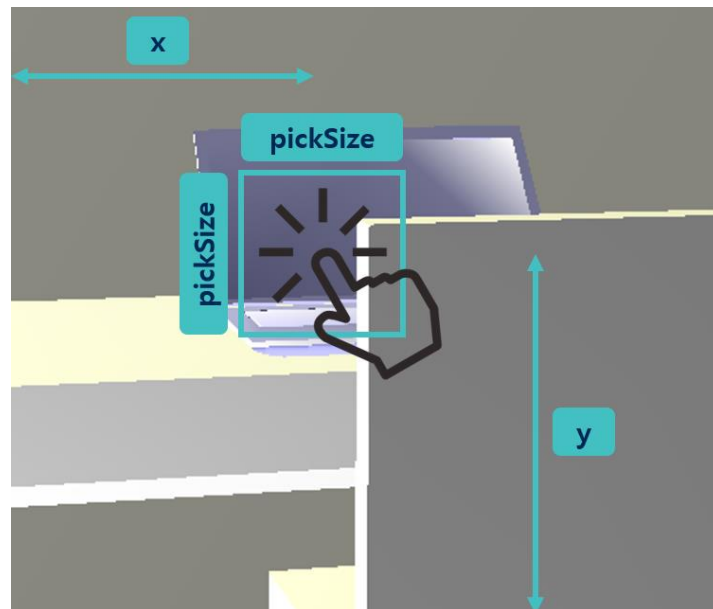


Figure 20) example of mouse clicking with `pickSize`

Finally, we will draw all objects with `GL_SELECT` mode. By this step, the vertices with object name are stored to select buffer. But this step won't affect to render process. Then, OpenGL can distinguish each object but can't calculate z-depth. So, we will solve this problem.

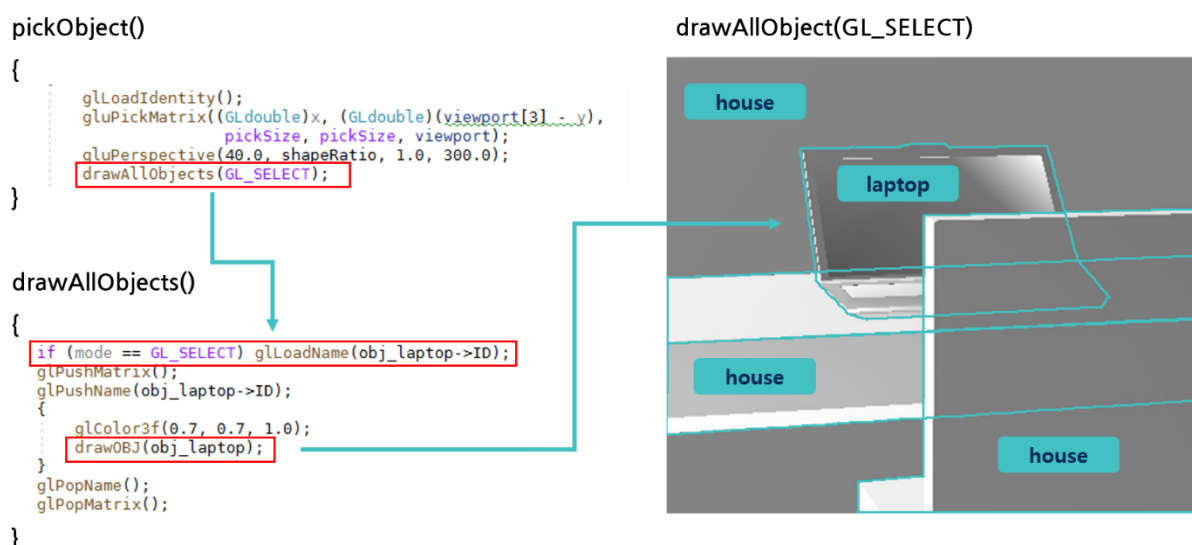


Figure 21) relation mouse clicking and drawing objects with select mode

### 4-3. Object Hits

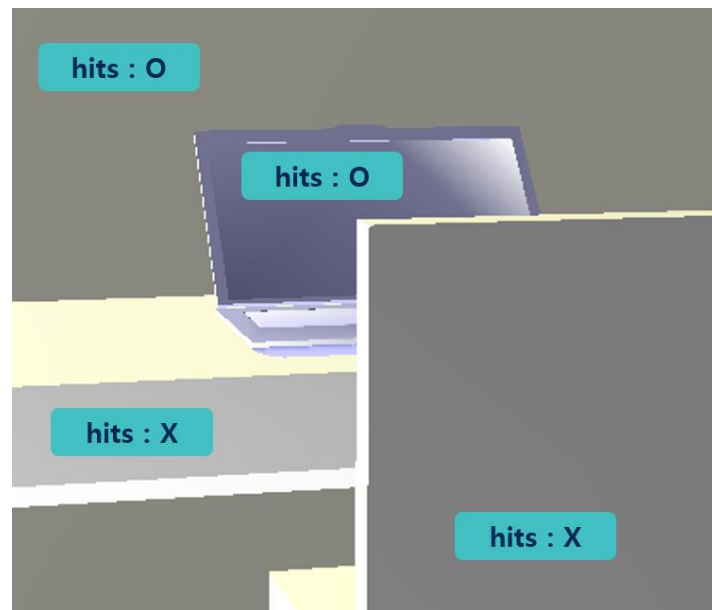


Figure 22) example of objects with hit process

After mouse pickObject() function, mouse click will generate the hit counts for counting object hits. This step is implemented by processHits(hits, selectBuf) with hits = glRenderMode(GL\_RENDER). At this example, the OpenGL can get two objects that names are laptop and house.

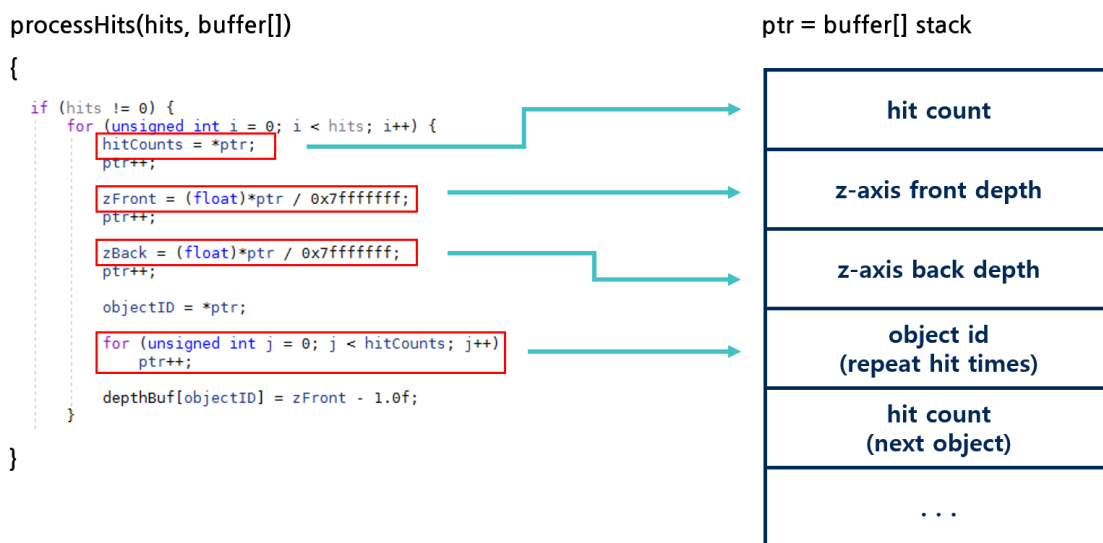


Figure 23) overall stack pointer of selectBuf as ptr

For this step, the selectBuf will stored stack values at ptr. And we defined each ptr values as above image. For this example, the mouse click generated two hits as laptop and house. As well, the objects are 3D-dimension not 2D-dimension. So, they have two z-depth values which are zFront and zBack by z-axis volume.

```
processHits(hits, buffer[])
{
    depthBuf[objectID] = zFront - 1.0f;

    for (int i = 0; i < bufferSize; i++) {
        if (zMin >= depthBuf[i]) {
            zMin = depthBuf[i];
            selectionID = i;
        }
    }
}
```

Figure 24) compare z-depth values

But we will only be interested in z-front depth for comparing z-depth values. So, depthBuf[objectID] will get only z-front value by  $zFront - 1$ . The zFront value is less than integer 2. That why we subtract with 1 to zFront for easy comparing. The minimum of z-depth as zMin is the value that the minimum value of z-front. By this step, we can get not only the object ID which is nearest object but also each zFront values as below.

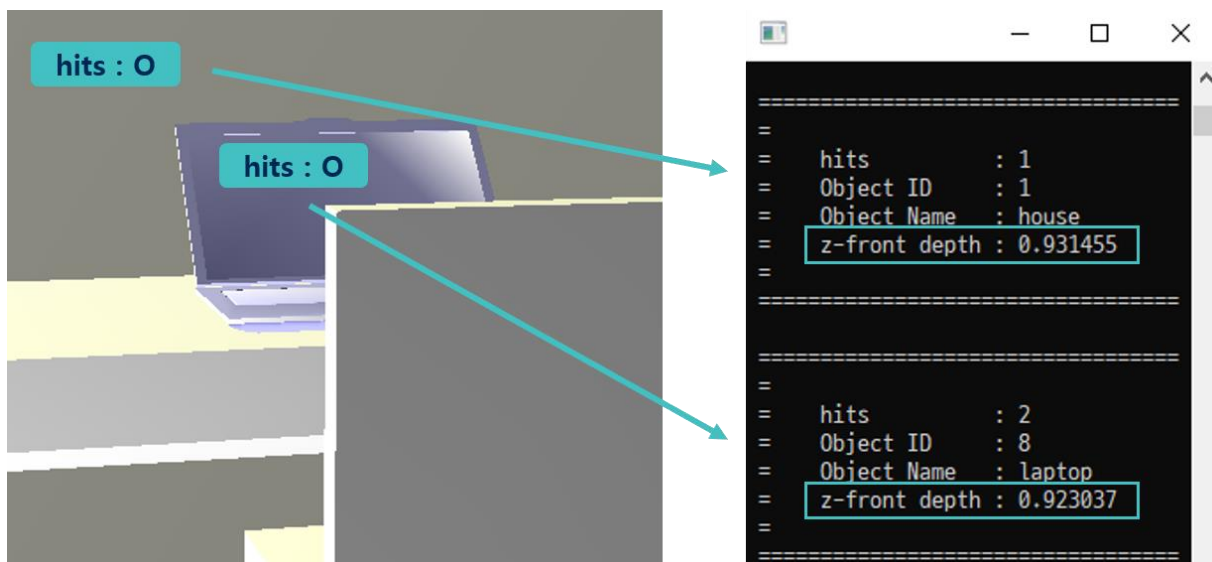
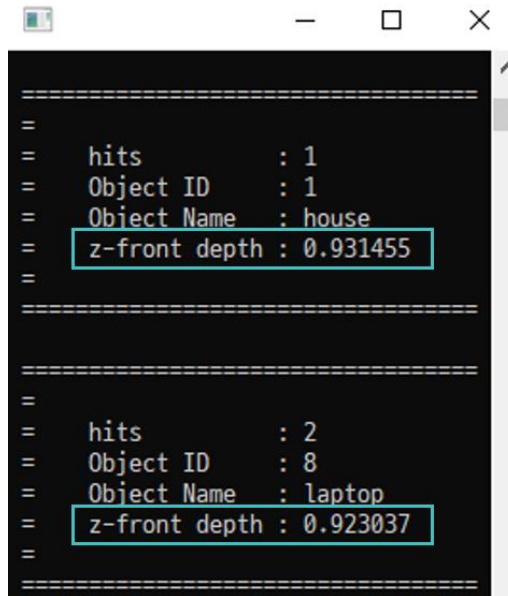


Figure 25) result of hit process with z-front depth value



```
=====
=
= hits          : 1
= Object ID     : 1
= Object Name   : house
= z-front depth : 0.931455
=
=====

=====
=
= hits          : 2
= Object ID     : 8
= Object Name   : laptop
= z-front depth : 0.923037
=
=====
```

Figure 26) result of zMin comparing

Finally, we can get the nearest object by comparing zFront values and achieve the distinction what is selected object by mouse clicking. At this example, the selected object is object->ID = 2 as object\_laptop.

## 5. Conclusion

Through this project, it was a good experience to apply skills we have learned in this course. In addition, we could learn several new knowledges such as text printing or passive mouse control. Although our project ended here to meet the deadline, it still has plenty of possibility to be developed more.

For example, background music or sound effect can be added. We can also improve move algorithm by collision detection since the current code is hard-coded. It would be a better project if we make an improved version of the game by supplementing these points in the future.

## 6. Contribution

김동훈 – basic coordinate calculation, viewport, passive mouse, trackball, object offset, door rotating

이승원 – password algorithm, obj loader, locating object by blender, text print, fixed moving area

정하빈 – generate VAO, objects rendering, pick object by mouse, object selection & deselection, lighting

## 7. References

- [1] free modeling : <https://free3d.com>
- [2] object house (house, door, bed, etc.) : <https://www.turbosquid.com/FullPreview/Index.cfm/ID/1321186>
- [3] object house (sofa, cushion, etc.) : <https://free3d.com/3d-model/stranger-things-byers-house---low-poly-702510.html>
- [4] object house (desk, etc.) : <https://free3d.com/3d-model/simple-room-415120.html>
- [5] object books : <https://free3d.com/3d-model/books-paperback-v1--848619.html>
- [6] object numbers : [https://www.cgtrader.com/free-3d-models/number?file\\_types%5B%5D=51](https://www.cgtrader.com/free-3d-models/number?file_types%5B%5D=51)
- [7] text print :  
<https://stackoverflow.com/questions/18847109/displaying-fixed-location-2d-text-in-a-3d-opengl-world-using-glut>
- [8] font rendering : <http://freeglut.sourceforge.net/docs/api.php#FontRendering>
- [9] obj loader : <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/>
- [10] VAO & VBO : <https://mr-dingo.github.io/opengl/2019/01/02/VAOandVBO.html>
- [11] OpenGL vertex Buffer Object : [http://www.songho.ca/opengl/gl\\_vbo.html#create](http://www.songho.ca/opengl/gl_vbo.html#create)
- [12] OpenGL IBO :  
<https://huiyu.tistory.com/entry/OpenGL-IBO%E5%BC-%EC%82%AC%EC%9A%A9%ED%95%9C-%ED%81%90%EB%B8%8C-%EA%B7%B8%EB%A6%AC%EA%B8%B0>
- [13] OpenGL Programming Guide – Selection and Feedback :  
<https://www.glprogramming.com/red/chapter13.html>
- [14] [OpenGL Tutorial] Selection : <http://www.gisdeveloper.co.kr/?p=53>
- [15] Lighting & Material : <https://skyfe79.gitbooks.io/opengl-tutorial/chapter10.html>
- [16] Lighting & Material Initialization :  
<http://www.devpia.com/MAEUL/Contents/Detail.aspx?BoardID=50&MAEULNo=20&no=821090&ref=821090>
- [17] OpenGL select distinction with object : [http://m.blog.daum.net/toyship/112?tp\\_nil\\_a=1](http://m.blog.daum.net/toyship/112?tp_nil_a=1)

[18] Graphics UI GLUT Fonts :

<http://haddock.stackage.org/lts-2.9/GLUT-2.7.0.1/Graphics-UI-GLUT-Fonts.html#t:BitmapFont>

[19] Survey of OpenGL Font Technology : <https://www.opengl.org/archives/resources/features/fontsurvey/>

[20] Lighting Basic : <http://soen.kr/lecture/library/opengl/opengl-9.htm>

[21] OpenGL Programming Guide – Lighting : <https://www.glprogramming.com/red/chapter05.html>

[22] glutSetCursor : <https://www.opengl.org/resources/libraries/glut/spec3/node28.html>