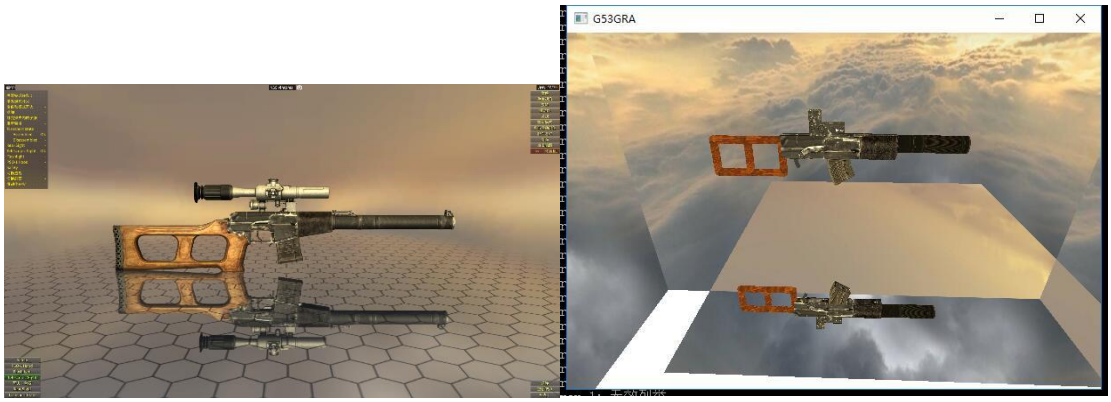
Computer Graphics | Coursework 2

Jinhao Wang 4302178

Introduction

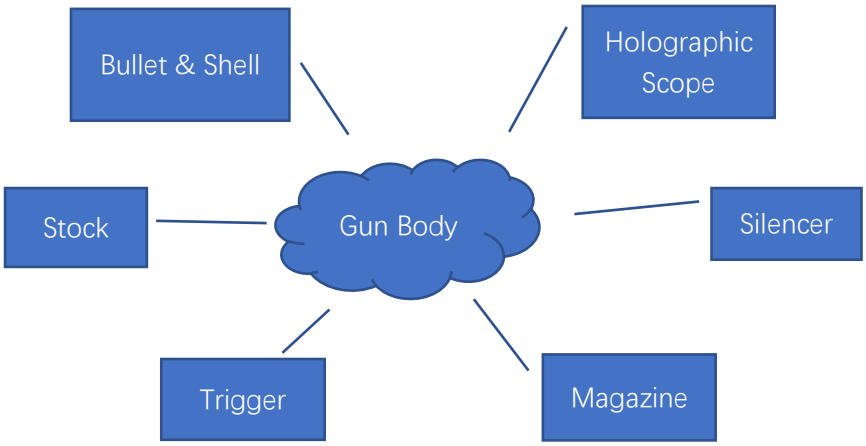
In this coursework, I was inspired by a game called World of Guns: Gun Disassembly. I made a raw model of VSS Vintorez (1987, Tula Arms Plant).



World of Guns: Gun Disassembly Coursework 2

Hierarchical Modelling

The gun model is constituted of multiple components:



Each component is constituted of multiple basic OpenGL models:

Gun Body: Front side (4 quads); Top side (1 quad); Back side (1 quad); Bottom side (1 quad); Left side (1 quad); Right side (1 quad).

Silencer: One cylinder represents the body (20 quads and 2 cones); 2 cones represent the bore.

Stock: 7 cylinders (20 quads); 4 spheres used at the corner.

Magazine: Curved front side (12 quads); Top side (1 quad); Curved back side (12 quads); Bottom side (1 quad); Curved left side (12 quads); Curved right side (12 quads).

Trigger: Same as the Magazine object.

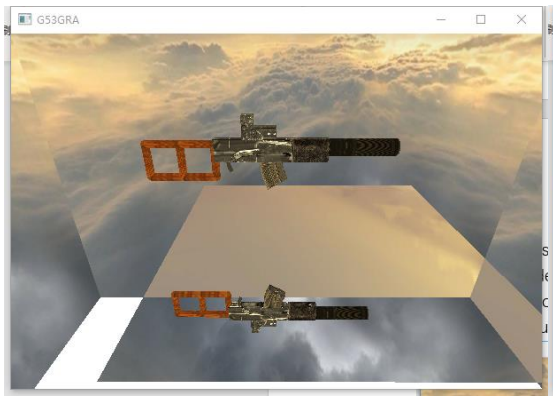
Holographic Scope: Front side (2 quads); Top side (2 quads); Back side (2 quads); Bottom

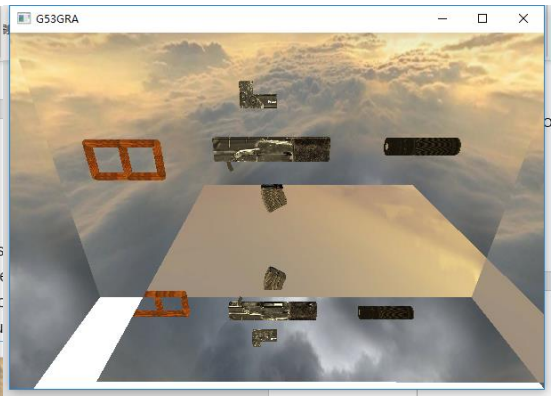
side (1 quad); Left side (1 quad); Right side (1 quad); Optical sight (2 semi-transparent quads).
Bullet: One cylinder represents the body (20 quads and 1 cone); One Cone represents the head;
Shell: One cylinder represents the body (20 quads and 1 cone).

Animation

I made 2 animation process in the coursework which can be controlled by users.

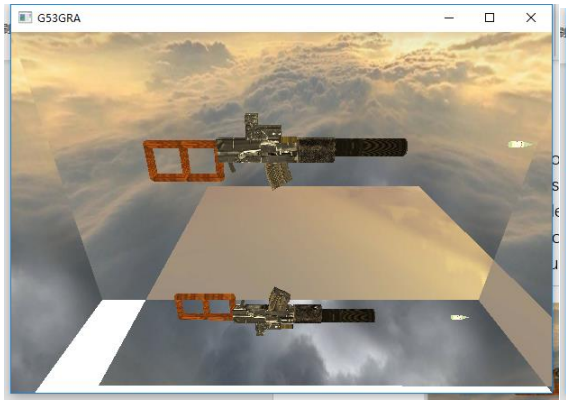
The first one is Assembling and Disassembling the gun. By pressing 'm', the gun will go into disassembly mode. By pressing 'n', the gun will return to assembly mode. Be advised, during the animation process, other inputs, which affect the gun itself, will not be triggered. Users cannot fire the gun in disassembly mode.

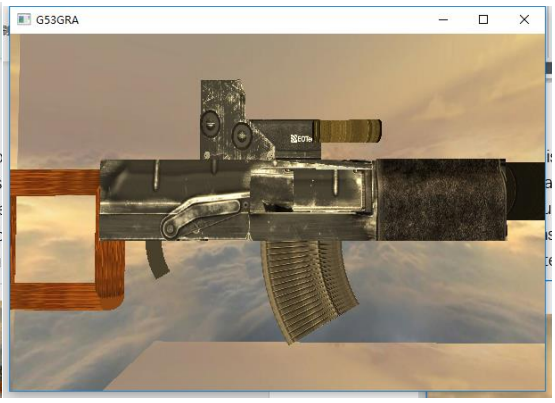




Assembly Mode Disassembly Mode

The second one is firing the gun. By pressing 'z', the gun will fire a bullet and throw out a shell. The trigger and the bolt will move backwards and forwards, imitating a real firing process. The speed is about 50 times slower than the normal. This animation cannot be triggered when the gun is in disassembly mode or already in a firing process. The bullet and shell will reset their positions after 10 seconds. Then the user can trigger the next firing process.





Firing the bullet and throwing the shell

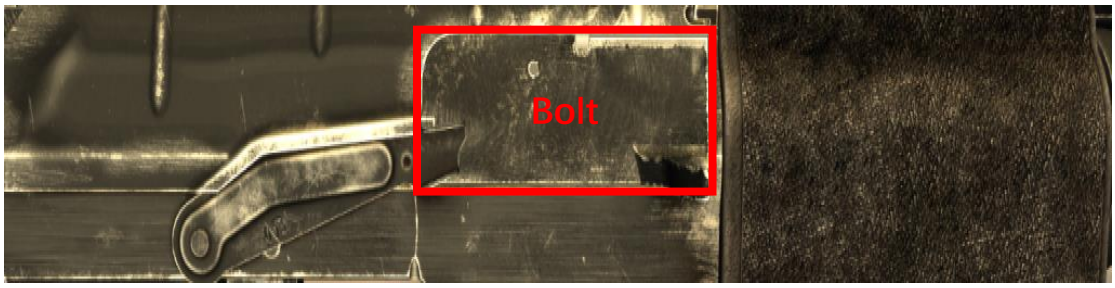
Texturing

In the coursework, all objects use images to make them look real. All images are screenshot from World of Guns, which can be checked in the Textures folder. Positions of texture are calculated and combined precisely to fit in the raw model.

For example, the bolt object, which used to throw out the shell, and the gun body object's front side use the same texture image 'body.bmp'. The textures of both are combined seamlessly as the original image.

Normal Mode

Firing Mode



Original Image

Lighting

In the scene, GL_LIGHTING is disabled in default. User can enable or disable it by pressing 'o' and 'p'.

When GL_LIGHTING is enabled, a positional light source will be enabled in the scene. User can switch the light colour between yellow (button '1'), red (button '2'), green (button '3'), and blue (button '4').

Viewing

The framework has already set a movable camera for users. I added two more key input. 'q' for moving upwards and 'e' for moving downwards.

The Mirror

As previously shown in the Introduction section, World of Guns has a plane underneath

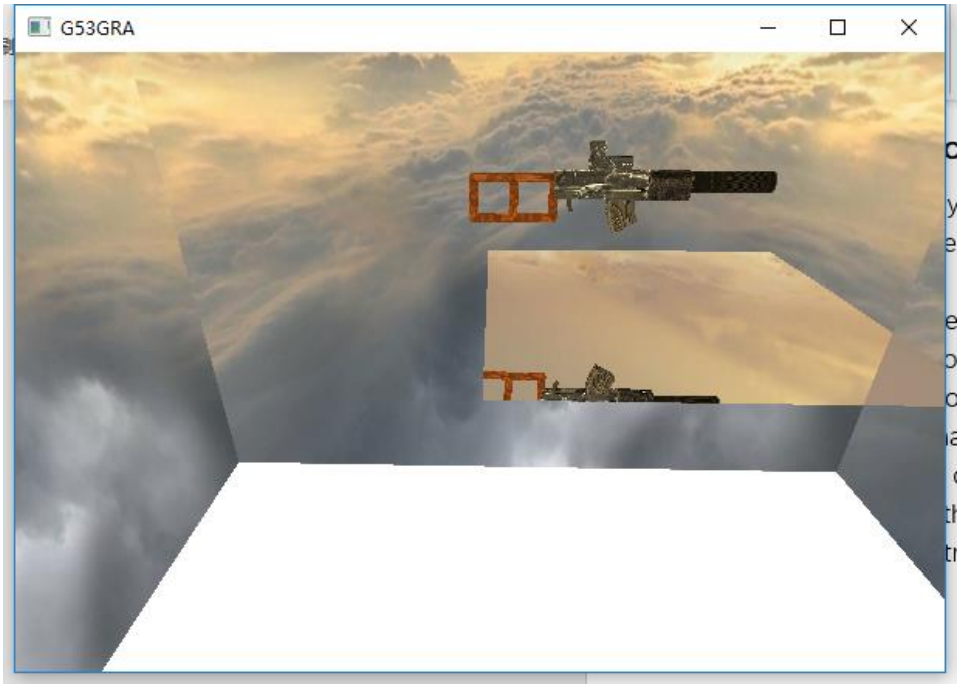
reflecting everything in the scene. Therefore, I decided to make a plane functioning as a mirror.

However, the actual work is not that easy. We have not covered deeply in this area in the class. I had to work it out on my own. By referencing different technical websites, I found that there are two main ways to do reflections. The first one is creating a mirroring camera, and texturing what the mirroring camera sees onto the mirror plane. The second one is drawing reflection objects and setting their stencil value. Only when objects' stencil value equals to the mirror's (which means, user is looking at those objects through the mirror), OpenGL will render those objects (reflections) in the corresponding area (mirror).

I chose the second approach in the end, since it is more understandable to me and easier to be implemented in the framework's Scene.cpp file.

```
//set stencil value; when drawing mirror, set reflection area stencil value to be 1
glClearStencil(0);
glClear(GL_STENCIL_BUFFER_BIT);
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

//Disable writting into the depth buffer, otherwise the buffer will be updated and the reflection cannot be drawn
glDepthMask(GL_FALSE);
DrawMirror();
glDepthMask(GL_TRUE);

//set stencil test, value not equal to the stencil value of mirror will not be drawn
glStencilFunc(GL_EQUAL, 1, 1);
glStencilFunc(GL_KEEP, GL_KEEP, GL_REPLACE);

//draw the reflection
glPushMatrix();
glScalef(1.0,-1.0,1.0);

for (DisplayableObject* obj : objects) {
```

Modification in Scene.cpp



The reflection objects are always existing in the scene. However, user cannot see them unless through the mirror area.