RTGP Project Report

BOURRY Amir

Project Details and Choices:

This project is a 3D graphics application built using OpenGL, GLFW, and GLEW, showcasing a first-person perspective within a detailed model of "the backrooms." The backrooms are a creepypasta from the internet. It is a fictional location. They are portrayed as an impossibly large extradimensional expanse of empty rooms. The project integrates various advanced rendering techniques, including bloom effects for dynamic lighting, a retro VCR pause effect, and interactive wall painting inspired by Splatoon. The main objective is to implement a 3D Application where the user can interact with the world and see light effects.

The application features a high dynamic range (HDR) effect achieved by bloom shaders, which increase the visual realism by emphasizing bright light sources. The VCR pause effect, adapted from a Shadertoy shader, introduces visual noise and distortion when the game is paused by pressing the ESC key. Additionally, the interactive painting mechanic allows users to simulate paint splashes on walls by rendering-colored circles on the surfaces the balls touch, which allow us to make the user interacts with the world.

OpenGL is used for rendering, GLFW for window management and input handling, and GLEW for accessing OpenGL extensions. The project also utilizes GLM for mathematical operations and uses shaders written in GLSL for visual effects. The backrooms model, sourced online, provides an environment, while shaders from LearnOpenGL and Shadertoy form the basis for bloom and VCR effects.

Algorithms and Techniques:

The bloom effect, inspired by LearnOpenGL tutorials, gives bright light sources through a multi-step process. Initially, the scene is rendered to a texture, from which bright areas are extracted. These areas undergo a Gaussian blur to create a glow effect. The final image is produced by merging the blurred bright areas with the original scene, resulting in a

visually striking HDR effect that highlights the ceiling lights. This gives the effect that the lights of the backrooms are “lagging” and produce a “horror movie” effect. The ceiling lights have an effect that changes multiples times following a cycle, so the effect is permanent.

The VCR pause effect replicates the visual distortions of paused VCR footage using a shader from Shadertoy. This effect is achieved by applying horizontal and vertical jitter, color offsets, and desaturation to the framebuffer, activated when the Escape button is pressed. The shader helps us to improve our horror movie effect, it is mainly inspired by the game Outlast (horror video game) where the player’s view is the view of a camera he holds the entire game, that has VCR effects.

Inspired by the game Splatoon, the painting effect allows users to interactively "paint" walls. Instead of modifying textures, which is a complicated task we tried but couldn’t accomplish, colored circles are rendered at the position the ball hits when the user clicks, simulating paint splashes. This method allow us to interact with the world without changing the textures. The ball is simply thrown from the camera in its own direction and when it hits a wall of the backroom, a simple colored circle is rendered at the collision position.

Implementation Details:

The application initializes GLFW for window and context creation, followed by GLEW to ensure OpenGL extensions are loaded. Shaders are compiled and linked for various effects, and textures for the crosshair, pause overlay, and other elements are loaded. The backrooms model is imported with its own textures.

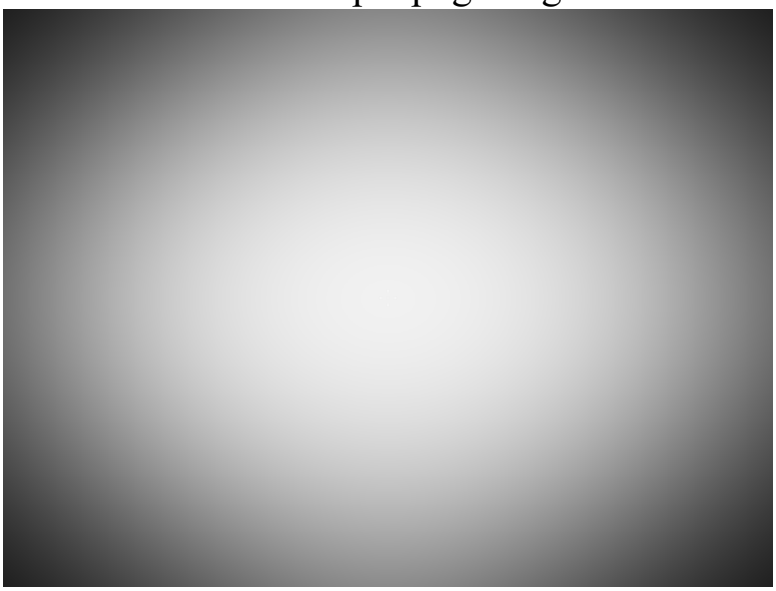
We use 5 shaders in the project.

```
Shader object_shader = Shader("shader.vert", "shader.frag");
Shader blur_shader = Shader("blur.vert", "blur.frag"); // bl
Shader bloom_shader = Shader("basic.vert", "bloom.frag"); //
Shader tex_shader = Shader("basic.vert", "tex.frag"); // sha
Shader pause_shader = Shader("basic.vert", "pause.frag"); //
```

The object shader refers as the world shader, it is the main one used to implement effects on the entire world. Then we find blur and bloom shaders for the effects. The bloom effects helps us for the lights, with it we

can obtain a dynamic lightning which is important for the horrific effect we want in the backrooms. The blur effect is used to create the Gaussian blur that affects the perception of the world. We use the tex shader which allows us to draw textures and then the pause shader we use to render the pause effect (VCR Camera effect on the world with the big PAUSE word written).

The crosshair is a simple png image :



We also use it to create a shadow effect on the 4 corners of the window. It adds the effect that the user is currently “running” or that his vision is obstructed, mixed with the light effect, we can recreate the feeling of an horror game.

The crosshair is rendered like this :

```
{
    glBindFramebuffer(GL_FRAMEBUFFER, pingpongColorbuffers[2]);
    glDisable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    tex_shader.Use();
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, crosshair);
    glUniform1i(glGetUniformLocation(tex_shader.Program, "tex"), 0);

    renderQuad();
}
```

It is also rendered after everything to avoid problems. We also disabled GL_DEPTH_TEST to ensure that the crosshair is drawn over everything (otherwise we won't see it). We also set the blending function for transparency to be sure that we can see the scene behind the crosshair. Then we simply apply the crosshair texture.

The main rendering loop handles input, updates the scene, and renders the visuals in multiple passes. Initially, the scene is rendered to a framebuffer. The bloom effect is then applied in a separate framebuffer, followed by combining the scene and bloom textures. The crosshair and pause overlay are rendered last, since they are the least important elements of the scene, and it helps to avoid problems of rendering. Input handling includes keyboard and mouse callbacks for camera movement and interaction, with specific logic for toggling the pause state and applying the VCR effect.

Camera movement is managed through keyboard and mouse inputs, controlling the first-person view. The pause functionality toggles the VCR effect with a key press, changing the visual state of the application. It also avoids that the user can move the camera when paused. The painting mechanic detects mouse clicks and renders paint splashes, permitting interactivity without modifying textures.

When the player clicks on his left mouse button, the Boolean mousepressed is set to true, so the following code is triggered :

```
if (currentFrame - lastbullet > 0.1) {
    glm::vec3 bulletPos = camera.Position + camera.Front * 0.5f;
    auto body = bulletSimulation.createRigidBody(SPHERE, bulletPos, glm::vec3(0.13f), camera.Front, 1.0f, 0.9f, 0.0f);
    // give bullet front speed
    body->setLinearVelocity(body->getLinearVelocity() + btVector3(camera.Front.x, camera.Front.y, camera.Front.z) * 20);
    spheres.push_back({body});
    lastbullet = currentFrame;
}
```

We ensure that the lastbullet is not too recent (to avoid sending to much bullet for one click only) and the bullet position is computed from the camera position. We give it a velocity and save it in the currentframe (to ensure no more bullets are sent in this frame). We then manage the bullet (which is a small sphere) :

```
bulletSimulation.dynamicsWorld->stepSimulation(min(deltaTime, maxSecPerFrame), 10);

std::vector<SphereData> newSpheres;
for (auto& sphere : spheres) {
    MyContactResultCallback callback;
    bulletSimulation.dynamicsWorld->contactPairTest(sphere.body, backroomBody, callback);
    if (callback.collided) {
        auto normal = glm::vec3(callback.normal.x(), callback.normal.y(), callback.normal.z());
        normal = glm::normalize(normal);

        // dot normal with bullet speed so we can check if the normal is flipped
        auto speed = sphere.body->getLinearVelocity();
        auto speedVec = glm::vec3(speed.x(), speed.y(), speed.z());
        if (glm::dot(normal, speedVec) > 0) {
            normal = -normal;
        }

        auto initial = glm::vec3(0, 0, 1);
        auto angle = glm::acos(glm::dot(initial, normal));

        auto axis = glm::normalize(glm::cross(initial, normal));
```

When there is a collision, the splat effect is drawn on the position of the collision. To draw it efficiently we compute the normal and then rotate the splat to point to normal to be sure it's on the surface and not inside or just half the splat. Once the splat is drawn and rotated to match the normal, we delete the bullet.

```
    glm::mat4 rot;
    if (angle > 0.001 && axis.x == axis.x) {
        rot = glm::rotate(glm::mat4(1.0f), angle, axis);
    } else {
        rot = glm::mat4(1.0f);
    }
    auto pos = glm::vec3(callback.position.x(), callback.position.y(), callback.position.z()) +
        normal * -0.04f;
    splats.push_back(SplatData{
        pos,
        rot});
    bulletSimulation.dynamicsWorld->removeRigidBody(sphere.body);
    delete sphere.body;
```

Since we have physics in our scene for the bullets, we chose to implement jump.

```
if (key == GLFW_KEY_SPACE && action == GLFW_PRESS) {
    playerBody->activate();

    playerBody->setLinearVelocity(playerBody->getLinearVelocity() + btVector3{0, 3, 0});
}
```

When the player presses space, the playerBody is activated and we give it a spam velocity on the Y axis so it make it jump a bit.

To stress the player and give him the feeling that he is loosing control, we also choose to move him even if he doesn't touch the keyboard.

```
playerBody->activate();
playerBody->setLinearVelocity(
    playerBody->getLinearVelocity() * btVector3{0, 1, 0} + btVector3{movement.x, 0, movement.z});
```

Obviously he is not moving on the Y axis (otherwise he will jump for nothing) but the effect of moving combined to the light effects enhance the horror effect.

Performance Evaluation:

Performance was measured in frames per second (FPS) under various conditions. During normal gameplay, the application maintains an average of 60 FPS on a mid-range GPU. Enabling the bloom effect reduces the frame rate to approximately 45-50 FPS due to the additional Gaussian blur processing. The VCR pause effect, however, has minimal impact on performance, maintaining 60 FPS even when active.

Optimization efforts focus on minimizing the performance impact of the bloom effect by refining the Gaussian blur shader and reducing the number of samples.

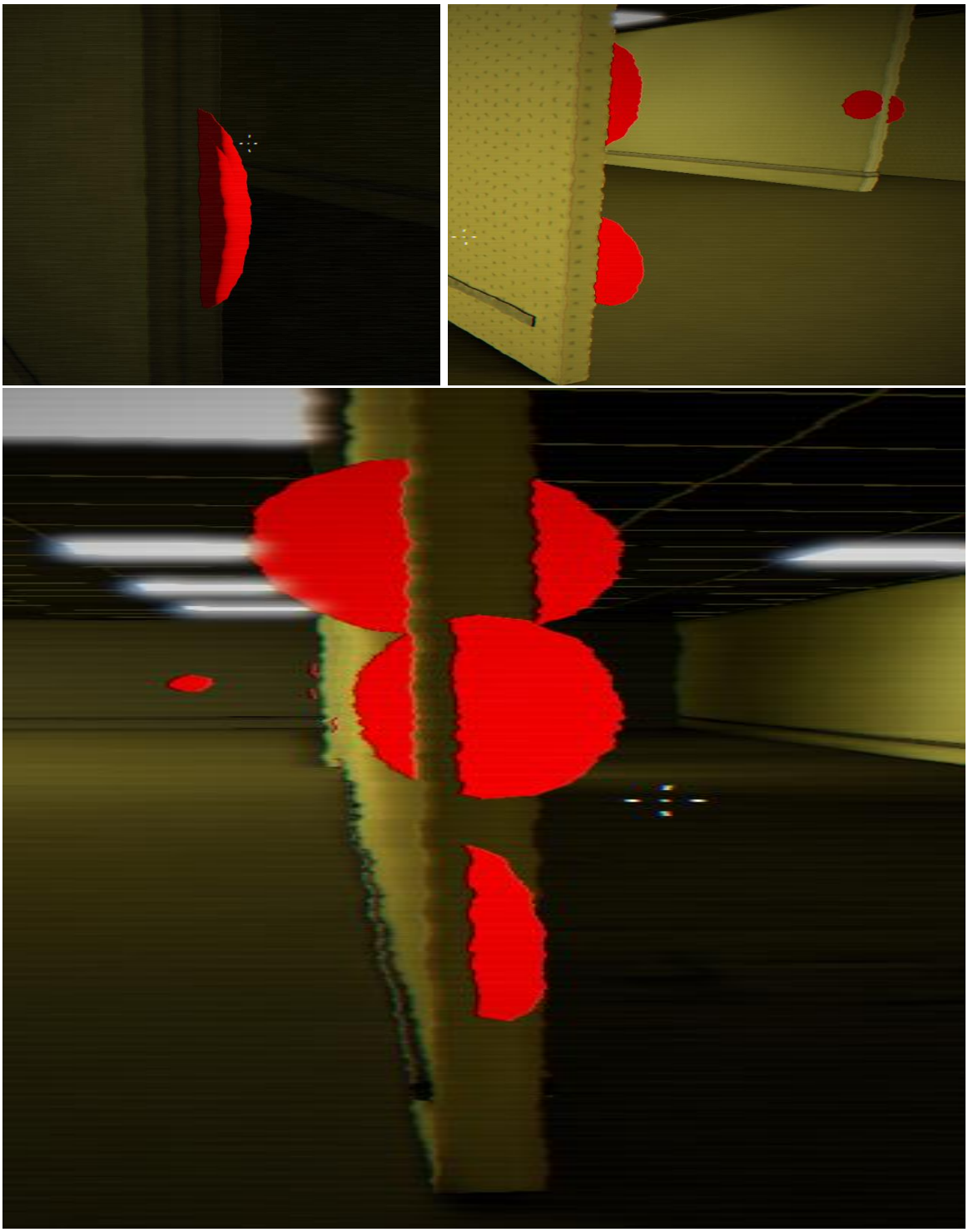
Performance evaluation was conducted on a system with an AMD Ryzen 7 6800H CPU, AMD Raden RX 6650M GPU, 32GB DDR4 RAM, and running Windows 11. This setup shows the application can deliver visual effects with good frame rate.

Problems in the project :

The project is not perfect, it is important to look on the problems and to think about ways to fix them. For now, 3 main problems are identified.

1) Splats don't render well on small surfaces

The splats are rendering well on the wall and the ground. But when the surface is not big (such as the side of a wall) a part of the splat is flying as shown in this screenshot:



The splats should not render for the part that is flying or render on the side wall. To fix it we should draw a square on the surface. Each part of the square that are not on a surface (so the part flying) should not be drawn. The lasting part of the square (so the one on the surface) then should be drawn red to give illusion that the splat is only on the surface and nowhere else.

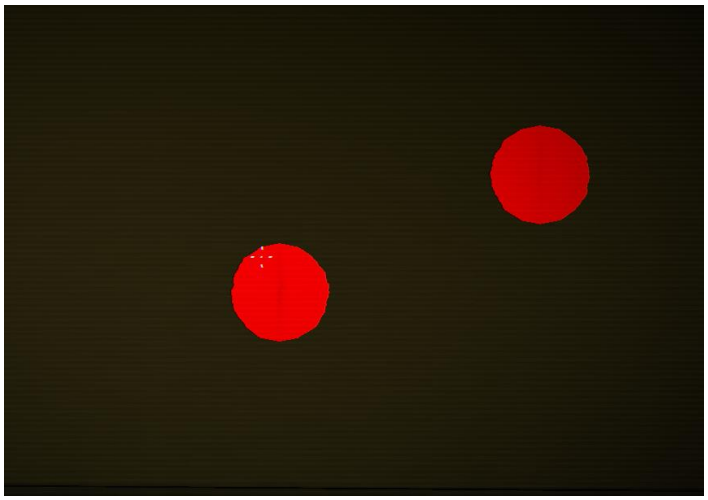
2) Bullets don’t always draw splats

For a reason we couldn’t find, the bullets doesn’t always draw splats on surfaces. The reason might be that the angle the bullets touch the surface is not good enough for our statement-based computation and the normal. Since the normal is not found, the splats simply don’t draw, while the bullet “did it’s job” so it is deleted.

A way to fix this would be to double-check if the splat is drawn successfully, if not, to draw it manually at the collision position by getting the normal of the surface.

3) Splats are affected by lightning effect separately which renders in a weird way :

Since the splats are not inside the surface texture but just drawn above the surface, they are not always affected by the lights effect the same way. For example, in this screenshot :



Splats on the wall are still lighted when the wall is completely in darkness. This is a problem, because when there are many splats and the room is dark, we still see them, which can decrease the feeling of horror game.

We have found these 3 problems. It will be interesting to search ways to implement the fixes we proposed above to fix them. The main problem is that the splats are not part of the texture. If it was, two of these problems will probably not occurs (splats flying and splats illuminated even in the dark room).

Conclusion:

This project successfully integrates rendering techniques in a real-time 3D graphics application where we can interact. The use of shaders for bloom and VCR effects, along with interactive painting, creates a good and engaging environment. Performance remains good, with potential for further optimization to enhance smoothness. Overall, the project showcases a horror version of Splatoon in the backrooms. We made some choices such as using a model for the rooms instead of drawing them by hand using Quads or render splashes instead of directly changing the techniques. Those choices are the result of research and tutorials during the development phase because some ideas we wanted to implement weren’t not every time possible in terms of optimization or development ability.