```
//texture loading
unsigned int cvnd = loadTexture(dmapMath[17].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int recep = loadTexture(dmapMath[3].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dvnd = loadTexture(dmapMath[6].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dwall = loadTexture(dmapMath[10].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int swall = loadTexture(smapMath[10].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int diffMap = loadTexture(dmapMath[0].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int specMap = loadTexture(smapMath[0].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dsofa = loadTexture(dmapMath[8].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int ssofa = loadTexture(smapMath[8].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dsofabase = loadTexture(dmapMath[9].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int ssofabase = loadTexture(smapMath[9].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dtable = loadTexture(dmapMath[2].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int stable = loadTexture(smapMath[2].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dplane = loadTexture(dmapMath[4].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int splane = loadTexture(smapMath[4].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dglb = loadTexture(dmapMath[0].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int sglb = loadTexture(smapMath[0].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dglbb = loadTexture(dmapMath[2].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int sglbb = loadTexture(smapMath[2].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dhx = loadTexture(dmapMath[12].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int shx = loadTexture(smapMath[12].c_str(), GL_LINEAR, GL_LINEAR, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int drw = loadTexture(dmapMath[13].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int srw = loadTexture(smapMath[13].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int hrw = loadTexture(dmapMath[5].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dfloor = loadTexture(smapMath[14].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int sfloor = loadTexture(smapMath[14].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int dtran = loadTexture(smapMath[16].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
unsigned int stran = loadTexture(smapMath[16].c_str(), GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
```

Figure 1 – Texture images loading

```
Cone cone = Cone();
unsigned int coneMap = loadTexture("kuet.png", GL_REPEAT, GL_REPEAT, GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
ConeWithTexture ghostTail(1.0f, 2.0, 20,
    glm::vec3(1.0f, 0.7f, 0.7f), // Ambient color
    glm::vec3(1.0f, 0.7f, 0.7f),  // Diffuse color
    glm::vec3(0.1f, 0.1f, 0.1f),  // Specular color
    32.0f,                       // Shininess
    coneMap,              // Diffuse texture
    coneMap              // Specular texture
);
```

Figure 2 – Cone and texture applying

```
    // Sample texture color
    vec3 textureColor = vec3(texture(material.diffuse, TexCoords));

    if (!anyLightEnabled) {
        // If no lights are enabled, make everything black
        FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    } else if (enableBlending) {
        // Blend logic: combine texture with material's surface color and lighting
        vec3 surfaceColor = material.ambient + material.diffuseColor * result + material.
        FragColor = vec4(textureColor * surfaceColor, 1.0);
    } else {
        // Use plain texture multiplied by lighting result
        FragColor = vec4(textureColor , 1.0);
    }
}
```

Figure 3 – Texture Color manual

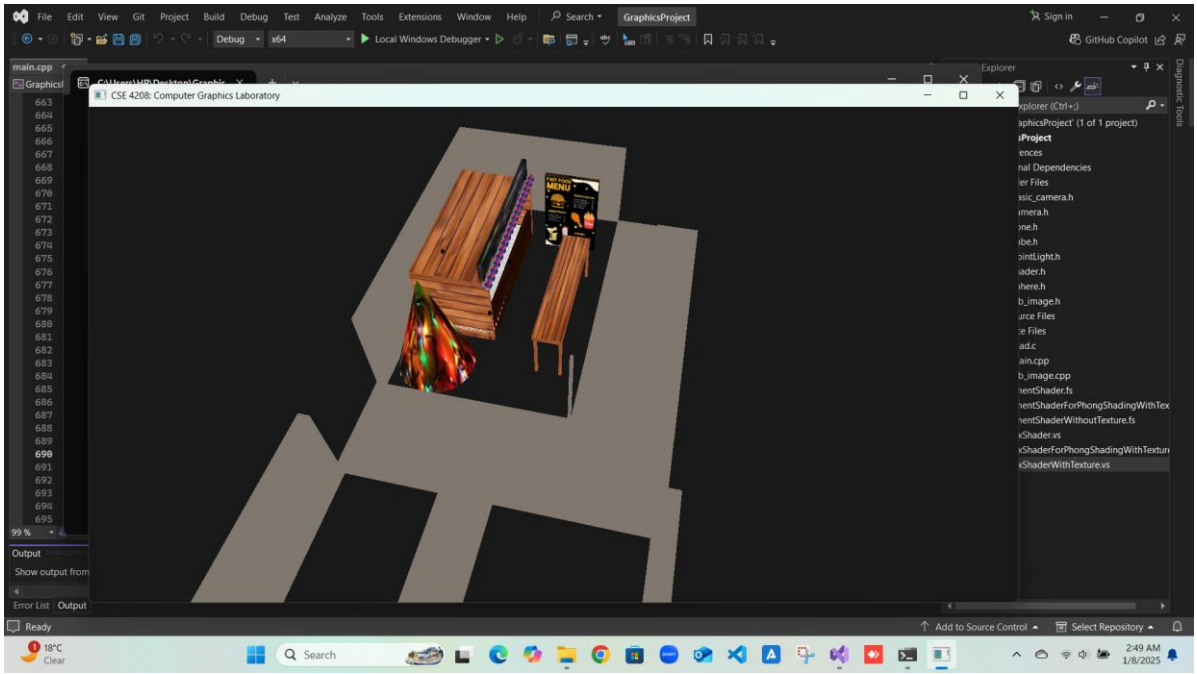

Figure 4 – Without texture “T” pressed

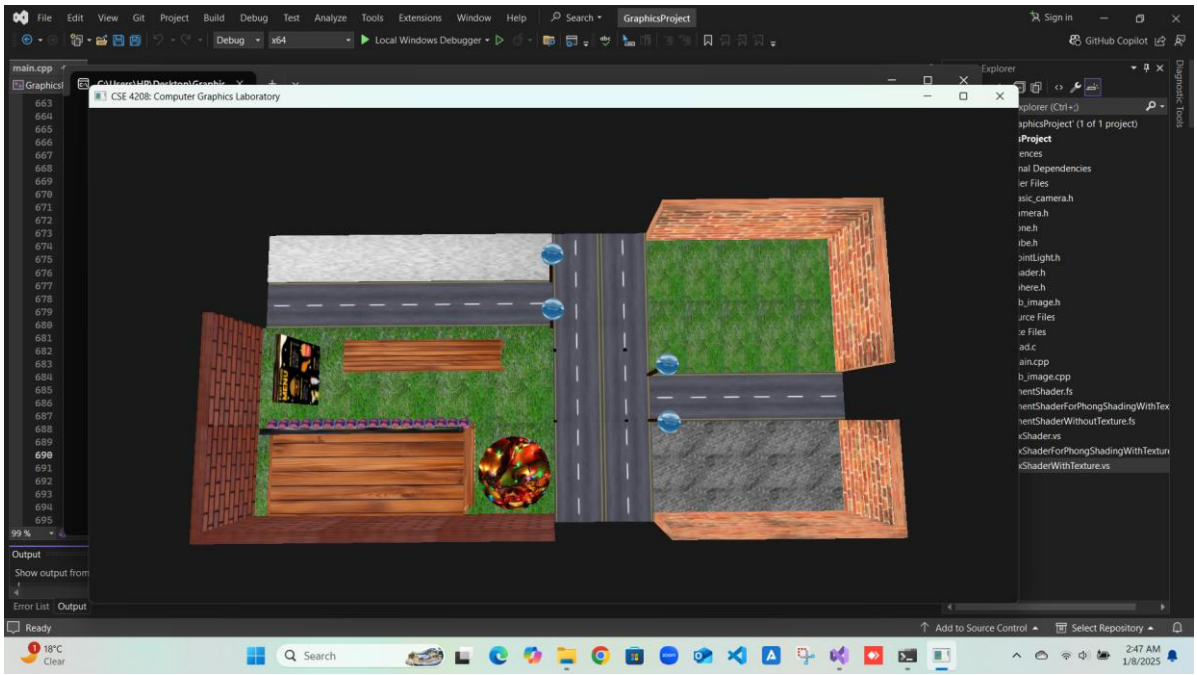

Figure 5 – Front View

```
// Function to calculate lighting effect from a point light
vec3 CalcPointLight(PointLight light, vec3 N, vec3 V, vec3 fragPos)
{
    vec3 L = normalize(light.position - fragPos);
    vec3 R = reflect(-L, N);

    float d = length(light.position - fragPos);
    float attenuation = 1.0 / (light.k_c + light.k_l * d + light.k_q * (d * d));

    vec3 ambient = material.diffuseColor * light.ambient;
    vec3 diffuse = material.diffuseColor * max(dot(N, L), 0.0) * light.diffuse;
    vec3 specular = material.specularColor * pow(max(dot(V, R), 0.0), material.shininess) * light.specular;

    return (ambient + diffuse + specular) * attenuation;
}
```

Figure 6 – Lighting effect apply

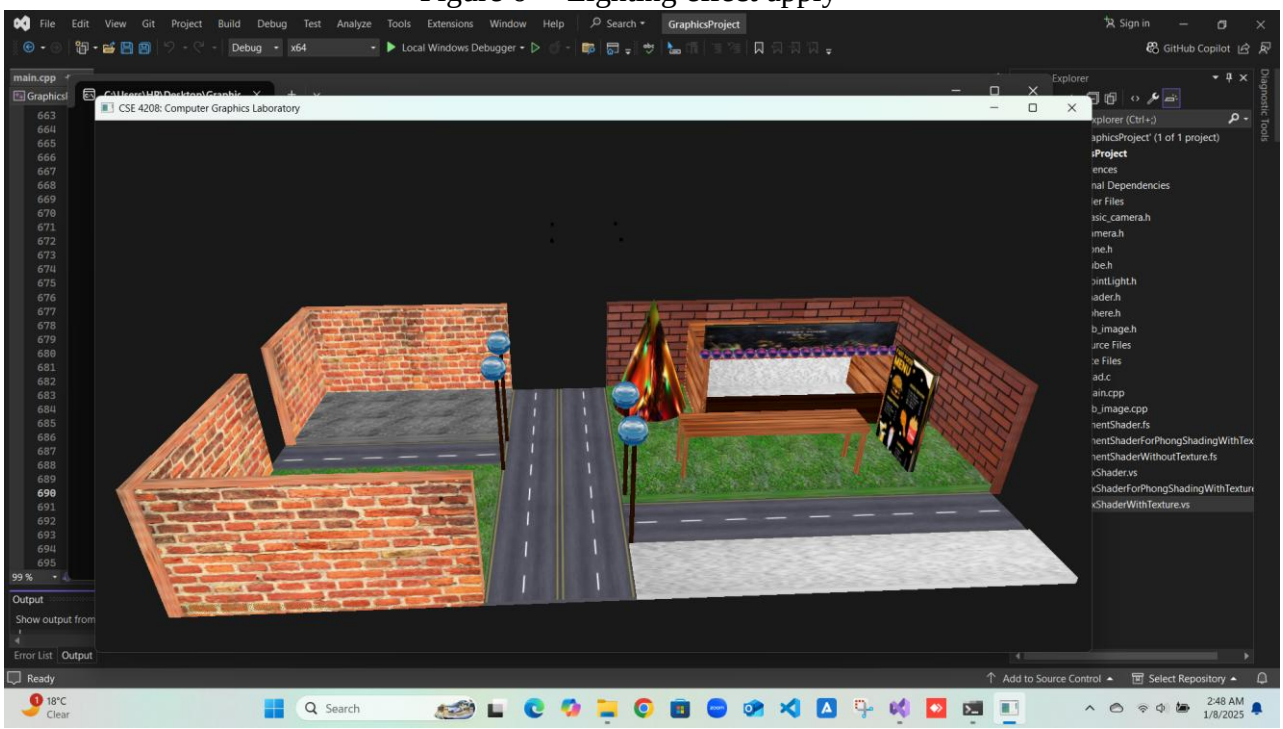

Figure 7 – Backward view

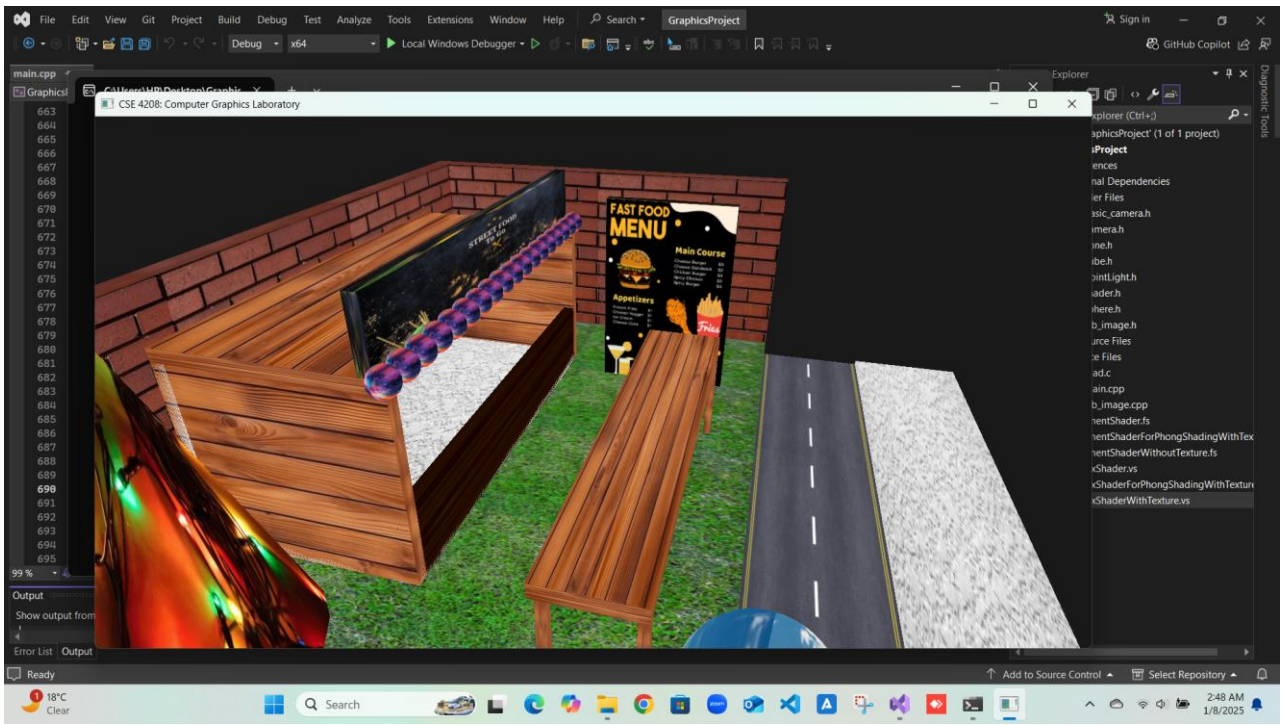

Figure 8 – Side view

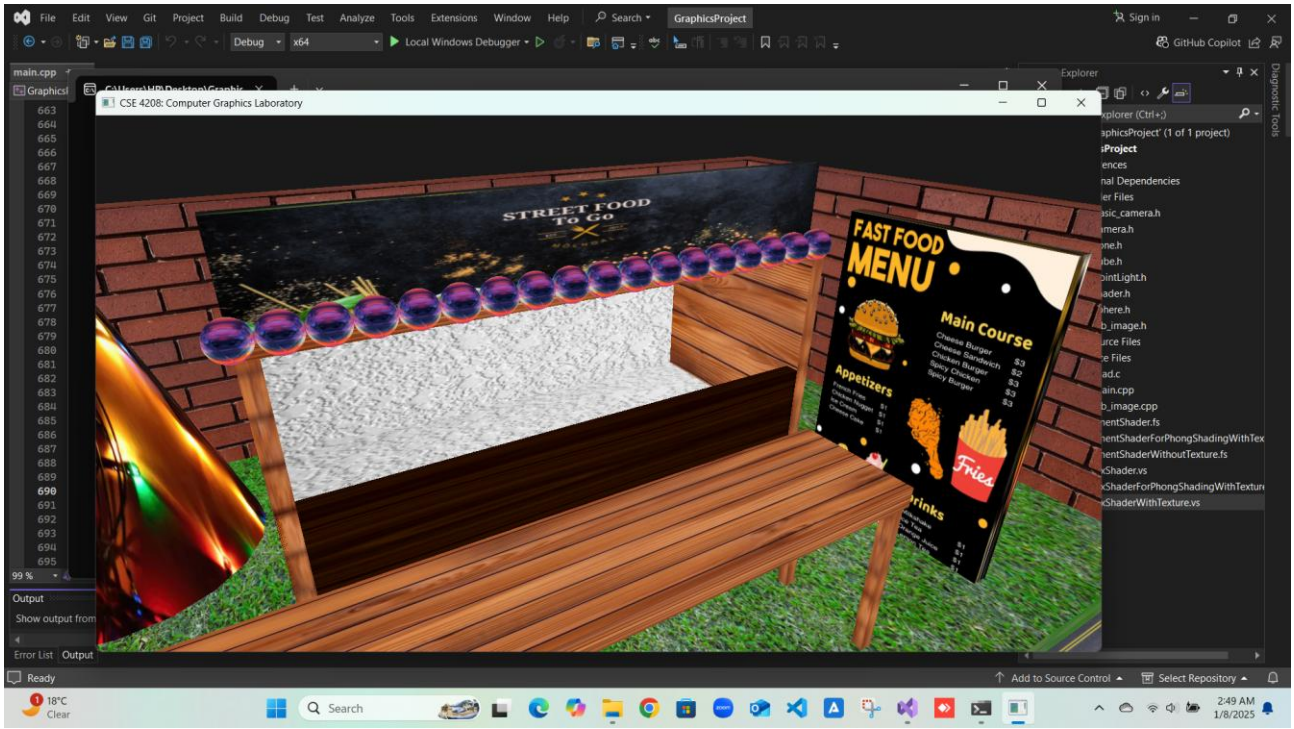

Figure 9 – Menu with texture

```
// Determine which shader to use based on blending mode
Shader* activeShader;


if (blendingEnabled) {
    activeShader = &fragmentBlendingShader;
    fragmentBlendingShader.use();
    fragmentBlendingShader.setBool("enableBlending", true);


}
else if (vertexBlendingEnabled) {
    activeShader = &vertexBlendingShader;
    vertexBlendingShader.use();
    vertexBlendingShader.setBool("enableVertexBlending", true);
}
else {
    activeShader = &lightingShaderWithTexture;
    lightingShaderWithTexture.use();

}


activeShader->setBool("pointLightEnabled[0]", pointOn1);
activeShader->setBool("pointLightEnabled[1]", pointOn2);
```

Figure 10 – Shader activating

```
if (glfwGetKey(window, GLFW_KEY_B) == GLFW_PRESS) {
    if (!bKeyPressed) { // If B was not previously pressed
        blendingEnabled = !blendingEnabled; // Toggle blending
        vertexBlendingEnabled = false; // Disable vertex blending
        bKeyPressed = true; // Mark B as pressed
    }
}
else {
    bKeyPressed = false; // Reset when B is released
}

// Handle toggling for vertex blending (V key)
if (glfwGetKey(window, GLFW_KEY_V) == GLFW_PRESS) {
    if (!vKeyPressed) { // If V was not previously pressed
        vertexBlendingEnabled = !vertexBlendingEnabled; // Toggle vertex blending
        blendingEnabled = false; // Disable fragment blending
        vKeyPressed = true; // Mark V as pressed
    }
}
else {
    vKeyPressed = false; // Reset when V is released
}
```

Figure 11 – Vertex and Fragment shader applied

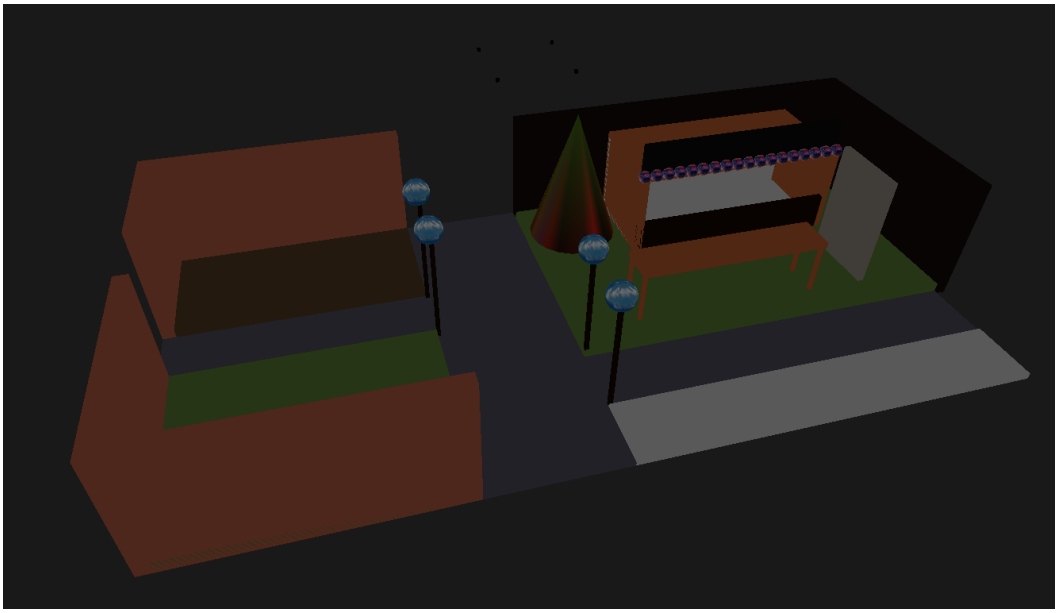

Figure 12 – Vertex Shader “V” pressed

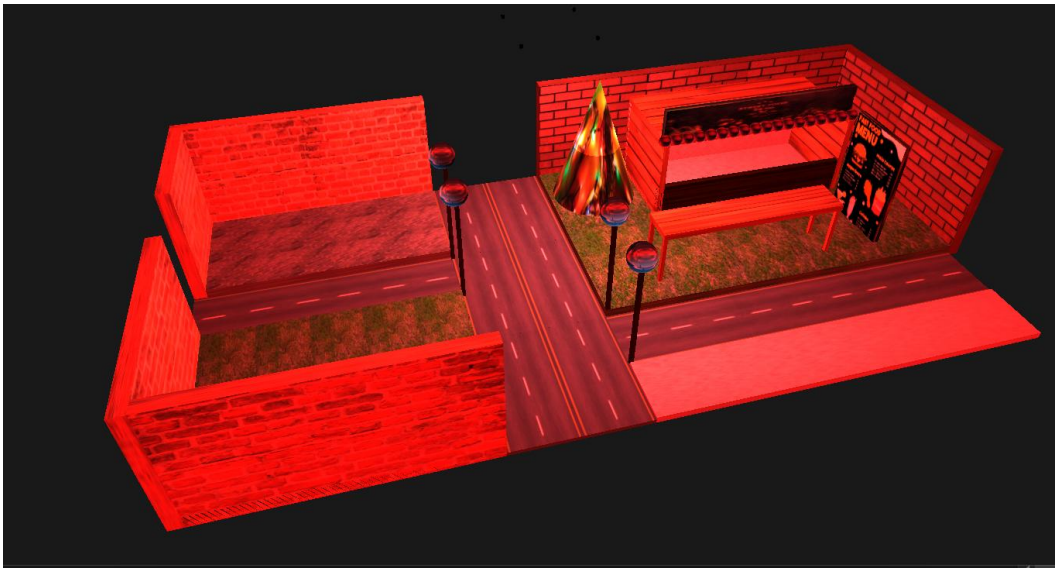

Figure 13 – Fragment Shader “B” pressed