Keith Donoghue 09607927

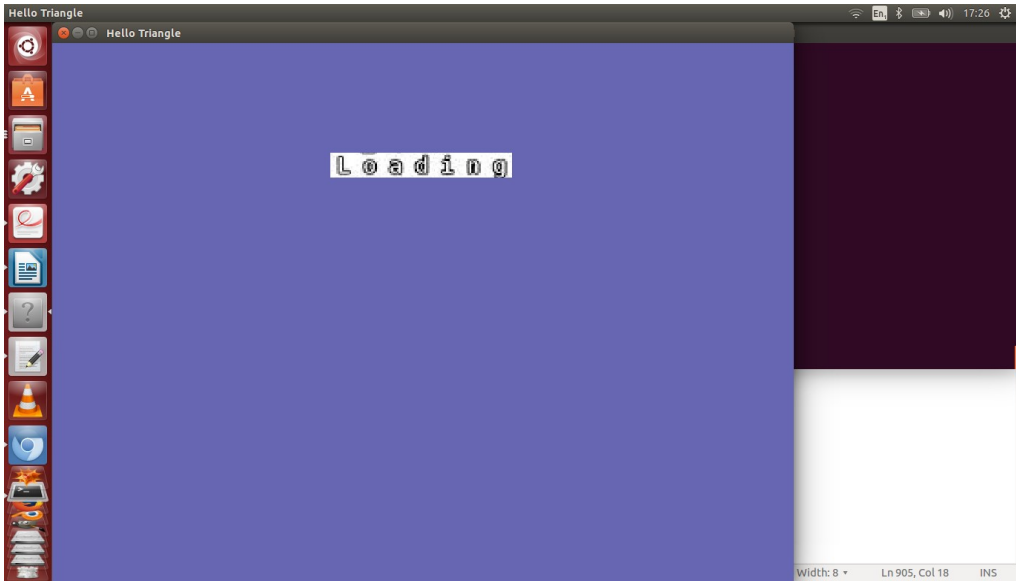
My game is a vehicle based military style game payers can choose to play as a land tank or as an aircraft. The object of the game is to shoot missiles at opposing vehicles, hopefully destroying them. There is a scoreboard in the corer of the screen which keeps track of the players remaining lives and also there “Kills”. The game starts with a welcome screen where players can start a game new game or get instuctions.

I implemented explosions manually. Collision detection I implemented using the bullet physics engine which i incorporated into the game.

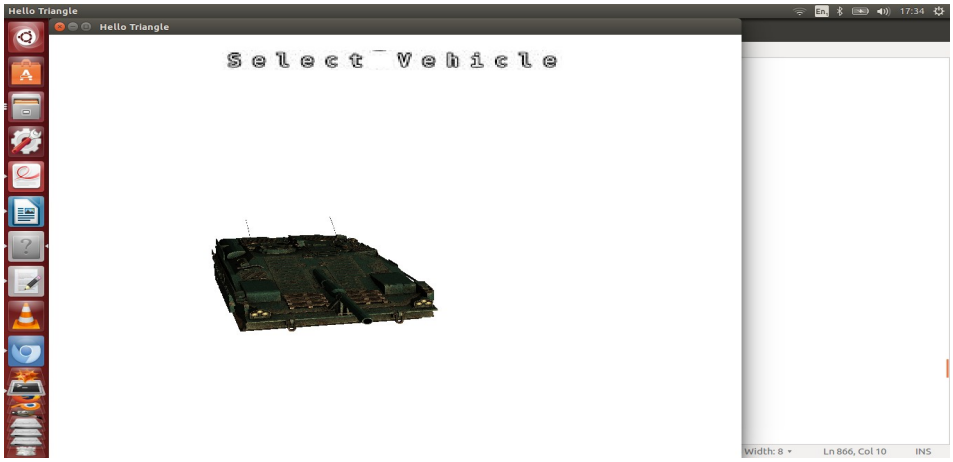
My start screen and choose vehicle screen were implemented using different display functions. I also have a loading screen whic displays a “Please Wait, Loading”, message while the models and textures are being loaded. The object of the game is to fire missiles at tanks on the ground which swerve and move. You can lock on to the tank with what i intended to be a ray casting technique however i nver got around to implementing it. The missiles do hone in on their targets however. So a implementing ray casting would complete the game. Allowing the layer to fire missiles directly forward is also an option which can already be done.

The main achievements in the game are the bullet phyics integration. And the fire synthesis. I downloaded images from the internet of smoke and flame textures. I then thresholded these to decide whether the alpha value was 1.0 or 0.0. I also for this portion of the code disabled depth testing. The collision detection algorithm tests for collisions and then activates an explosion at the point of the collision.

I downloaded bullet physics source code and installed in the program i create a global variable pointer called theWorld which points to all the information needed for the program to run. All position updates are handled through bullet. There is a loading screen while the textures and models are loading.



This is followed by a screen where you pick your vehicle,which rotates.



I have 3 shaders i the program 1 governs standard lighting with a blinn-phong shading model. The second coers billboarding in order to ensure the billboards used in the fire particle generator always face the camera.

As particles grow older different textures can be applied thus allowing fire to turn to dark smoke and then to light smoke

```
if (explosionList->life  > 900){
        glBindTexture(GL_TEXTURE_2D, textureID[6]);}

else {
        float f = ((float) rand() / (RAND_MAX));
        if (f < 0.2){
                glBindTexture(GL_TEXTURE_2D, textureID[8]);
                }
        else if (f < 0.4){
                glBindTexture(GL_TEXTURE_2D, textureID[9]);
                }
        else if (f < 0.6){
                glBindTexture(GL_TEXTURE_2D, textureID[10]);
                }
        else if (f < 0.8){
                glBindTexture(GL_TEXTURE_2D, textureID[11]);
                }
        else {  glBindTexture(GL_TEXTURE_2D, textureID[12]);
                }
        }

local1 = translate (local1, vec3 (0.0, 0, 15));

vec3 ambient(1.0f,1.0f,1.0f);
// update uniforms & draw
model = model1*local1;
glUniform3fv (Ambient_location,1,ambient.v );
glUniformMatrix4fv (proj_mat_location, 1, GL_FALSE, persp_proj.m);
glUniformMatrix4fv (view_mat_location, 1, GL_FALSE, view.m);
glUniformMatrix4fv (matrix_location, 1, GL_FALSE, model.m);
```

Some of the textures involved:





A video can capture the effect. The shader to ensure billboarding is s follows:

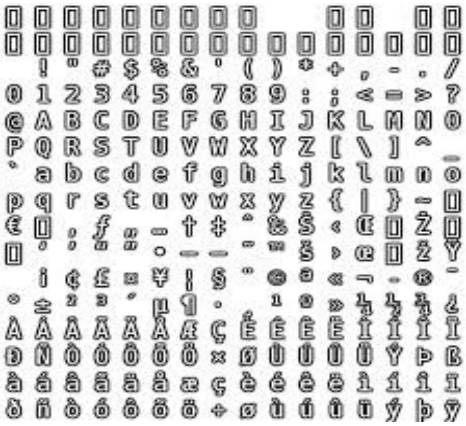
```
  // Convert position to clip coordinates and pass along\n\
mat4 modelview = view*model;\n\
for(int i = 0 ; i < 3; i++)\n\
        for(int j = 0 ; j < 3; j++)\n\
                {\n\
                        if (i == j)modelview[i][j]= 5; \n\
                        else modelview[i][j] = 0;}\n\
//gl_Position = vec4(20,20,20,1.0);\n\
 gl_Position =   vec4(proj *modelview*vertex_position);\n\
//  gl_Position =  vec4(vertex_position.x,vertex_position.y,0,1.0);\n\
//LightIntensity =  vertex_position;\n\
        UV = uvs ; \n\
```

The third ahadre is for the scoreboard and keeps a texture of ascii charachters at the front of all other geometry.

```
layout (location =0)in vec2 vertexPosition_screenspace;\n\
layout (location =1)in vec2 vertexUV;\n\
out vec2 UV;\n\
void main(){\n\
 \n\
   // Output position of the vertex, in clip space\n\
   // map [0..800][0..600] to [-1..1][-1..1]\n\
```

```
    vec2 vertexPosition_homoneneousspace = vertexPosition_screenspace - vec2(400,300); // [0..800][0..600] -> [-400..400][-300..300]\n\
    vertexPosition_homoneneousspace /= vec2(400,300);\n\
    gl_Position =  vec4(vertexPosition_homoneneousspace,-1,1);\n\
 \n\
    // UV of the vertex. No special space for this one.\n\
    UV = vertexUV;\n\
```

gl_Position = vec4(vertexPosition_homoneneousspace,-1,1); is the key line. The -1 keeps the text right to the fore. The following texture was used a implement a DrawString function to pass the vertx data to the buffer. The uv values are mapped for each charachter.



The plane and the camera move in sync with the plane positioned in wh feels natural although this can be altered.