```
...

GLFWwindow *window; // Main application window
string RESOURCE_DIR = "./"; // Where the resources are loaded from

GLuint progID;
map<string,GLint> attrIDs;
map<string,GLint> unifIDs;
map<string,GLuint> bufIDs;
int indCount;

...

// This function is called once to initialize the scene and OpenGL
static void init()
{
        ...
        //
        // Vertex buffer setup
        //

        // Load OBJ geometry    // Load OBJ geometry
        vector<float> posBuf;
        vector<float> norBuf;

        ... // Parse OBJ

        indCount = posBuf.size()/3; // number of indices to be rendered

        ...

        // Generate 2 buffer IDs and put them in the bufIDs map.
        GLuint tmp[2];
        glGenBuffers(2, tmp);
        bufIDs["bPos"] = tmp[0];
        bufIDs["bNor"] = tmp[1];

        // Send data to GPU
        glBindBuffer(GL_ARRAY_BUFFER, bufIDs["bPos"]);
        glBufferData(GL_ARRAY_BUFFER, posBuf.size()*sizeof(float), &posBuf[0], GL_STATIC_DRAW);
        glBindBuffer(GL_ARRAY_BUFFER, bufIDs["bNor"]);
        glBufferData(GL_ARRAY_BUFFER, norBuf.size()*sizeof(float), &norBuf[0], GL_STATIC_DRAW);

        glBindBuffer(GL_ARRAY_BUFFER, 0);
}

// This function is called every frame to draw the scene.
static void render()
{
        // Clear framebuffer.
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Get current frame buffer size.
        int width, height;
        glfwGetFramebufferSize(window, &width, &height);
        float aspect = width/(float)height;

        // Set up projection matrix (camera intrinsics)
        mat4 P;
        P = perspective((float)(45.0*M_PI/180.0), aspect, 0.01f, 100.0f);

        // Modelview matrix: camera and world
        float A[16];

        // Tell OpenGL which GLSL program to use
        glUseProgram(progID);
        // Pass in the current projection matrix
        glUniformMatrix4fv(unifIDs["P"], 1, GL_FALSE, &P[0][0]);
        // Enable the attribute
        glEnableVertexAttribArray(attrIDs["aPos"]);
        // Enable the attribute
        glEnableVertexAttribArray(attrIDs["aNor"]);
        // Bind the position buffer object to make it the currently active buffer
        glBindBuffer(GL_ARRAY_BUFFER, bufIDs["bPos"]);
        // Set the pointer -- the data is already on the GPU
        glVertexAttribPointer(attrIDs["aPos"], 3, GL_FLOAT, GL_FALSE, 0, (void *)0);
        // Bind the color buffer object to make it the currently active buffer
        glBindBuffer(GL_ARRAY_BUFFER, bufIDs["bNor"]);
        // Set the pointer -- the data is already on the GPU
        glVertexAttribPointer(attrIDs["aNor"], 3, GL_FLOAT, GL_FALSE, 0, (void *)0);

        // Send the modelview matrix and draw
        createIdentityMatrix(A);
        glUniformMatrix4fv(unifIDs["MV"], 1, GL_FALSE, A);
        glDrawArrays(GL_TRIANGLES, 0, indCount);

        // Unbind the buffer object
        glBindBuffer(GL_ARRAY_BUFFER, 0);
        // Disable the attribute
        glDisableVertexAttribArray(attrIDs["aNor"]);
        // Disable the attribute
        glDisableVertexAttribArray(attrIDs["aPos"]);
        // Unbind our GLSL program
        glUseProgram(0);
}

int main(int argc, char **argv)
{
        if(argc < 2) {
                cout << "Please specify the resource directory." << endl;
                return 0;
        }
        RESOURCE_DIR = argv[1] + string("/");

        // Set error callback.
        glfwSetErrorCallback(error_callback);
        // Initialize the library.
        if(!glfwInit()) {
                return -1;
        }
        // Create a windowed mode window and its OpenGL context.
        window = glfwCreateWindow(640, 480, "YOUR NAME", NULL, NULL);
        if(!window) {
                glfwTerminate();
                return -1;
        }
        // Make the window's context current.
        glfwMakeContextCurrent(window);
        // Initialize GLEW.
        glewExperimental = true;
        if(glewInit() != GLEW_OK) {
                cerr << "Failed to initialize GLEW" << endl;
                return -1;
        }
        glGetError(); // A bug in glewInit() causes an error that we can safely ignore.
        cout << "OpenGL version: " << glGetString(GL_VERSION) << endl;
        cout << "GLSL version: " << glGetString(GL_SHADING_LANGUAGE_VERSION) << endl;
        GLSL::checkVersion();
        // Set vsync.
        glfwSwapInterval(1);
        // Set keyboard callback.
        glfwSetKeyCallback(window, key_callback);
        // Set the mouse call back.
        glfwSetMouseButtonCallback(window, mouse_callback);
        // Set the window resize call back.
        glfwSetFramebufferSizeCallback(window, resize_callback);
        // Initialize scene.
        init();
        // Loop until the user closes the window.
        while(!glfwWindowShouldClose(window)) {
                // Render scene.
                render();
                // Swap front and back buffers.
                glfwSwapBuffers(window);
                // Poll for and process events.
                glfwPollEvents();
        }
        // Quit program.
        glfwDestroyWindow(window);
        glfwTerminate();
        return 0;
}

```

```
attribute vec4 aPos;
attribute vec3 aNor;
uniform mat4 P;
uniform mat4 MV;
varying vec3 vCol;

void main()
{
        gl_Position = P * MV * aPos;
        vCol = 0.5*(aNor + 1.0);
}
```