Goals and main task(s): Create renderers that will draw the previously-created graphical data

structures.

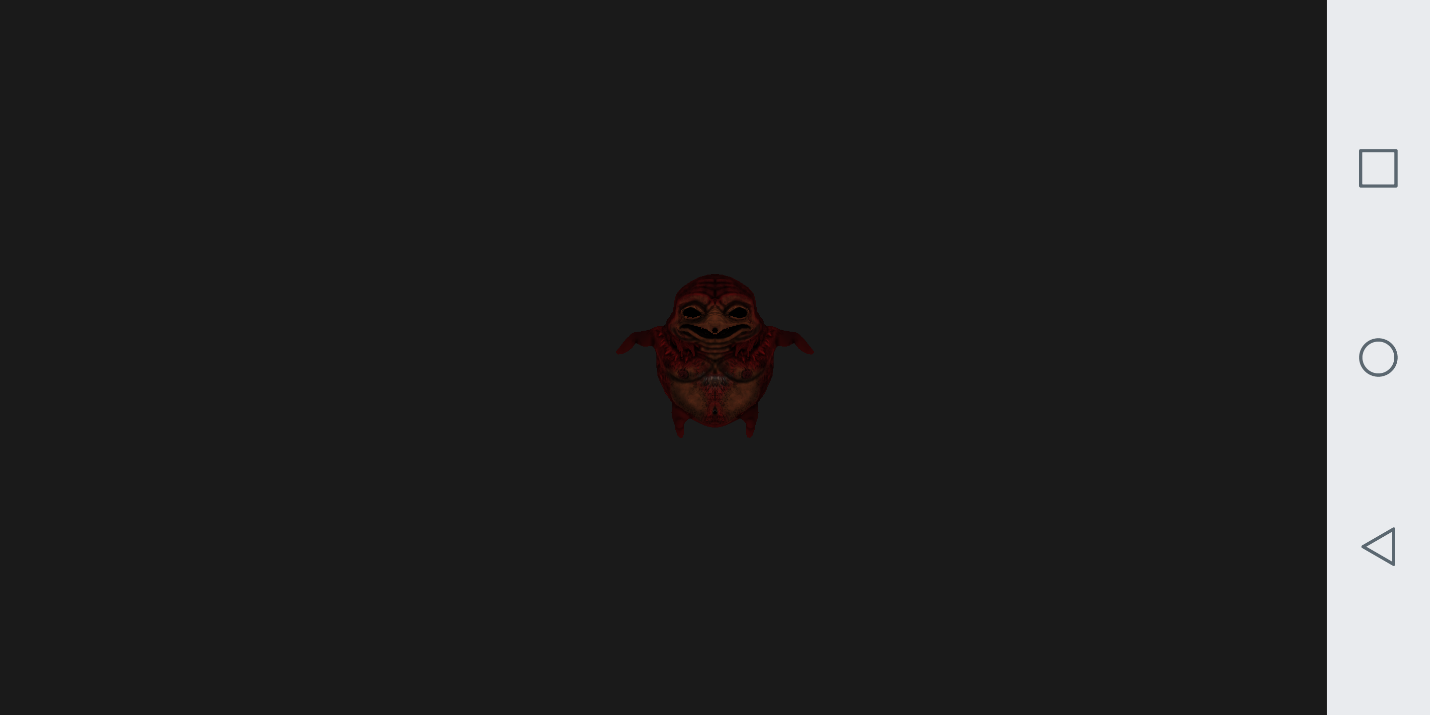
Tasks: Created forward renderer

Reflections: I created a forward renderer, which draws each mesh one-by-one and computes light calculations on the mesh. Thus the scene complexity of this renderer is: O(geometry \* lights). This renderer is the only three-dimensional renderer currently implemented. The code for its rendering process is shown below.

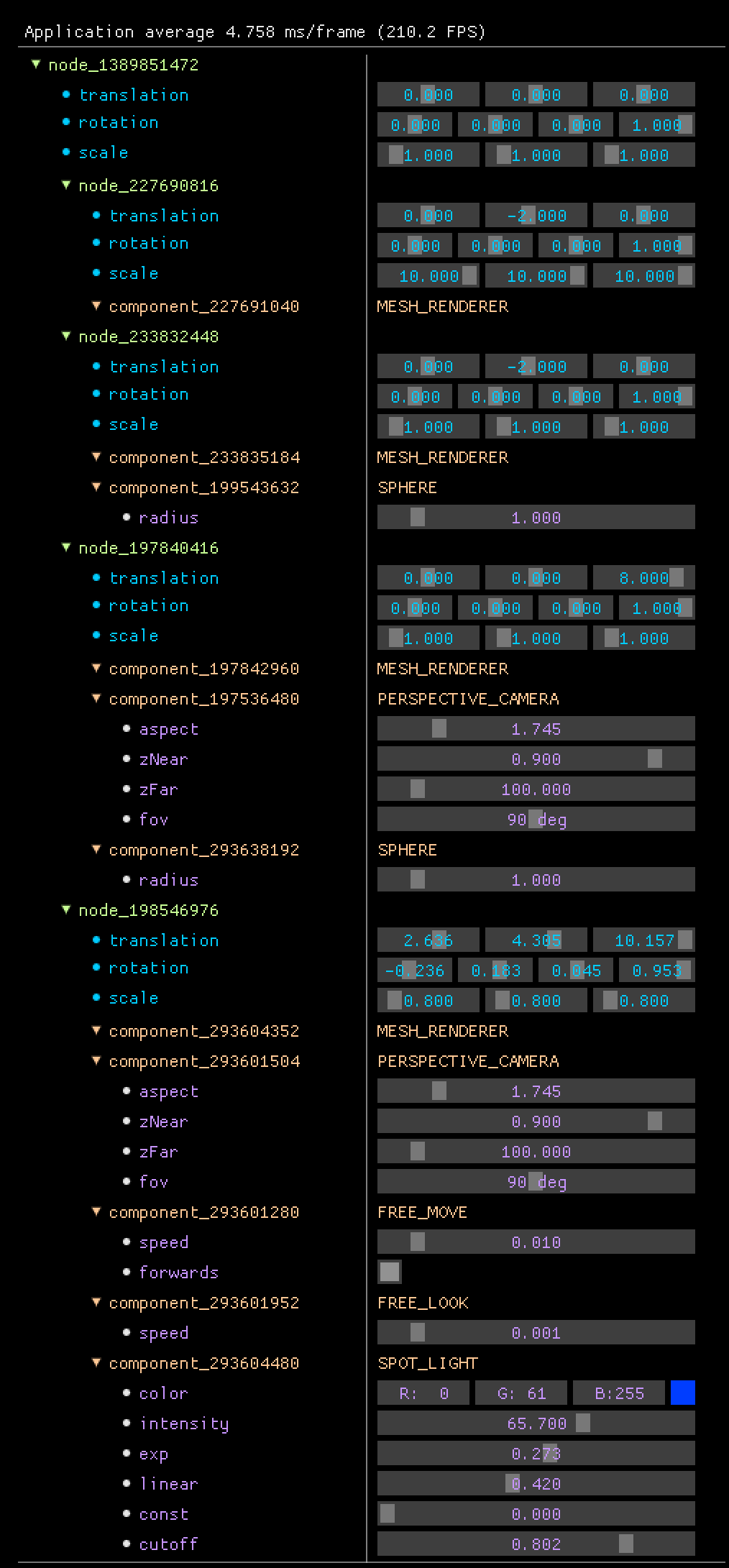
Tasks: Created GUI manager

Reflections: In addition to a three-dimensional graphics renderer, I also created a GUI manager that uses the imgui library to construct vertex buffers and then draw them to the screen. The GUI manager renders fonts and can display diagnostic information. The code for its rendering process is shown below.

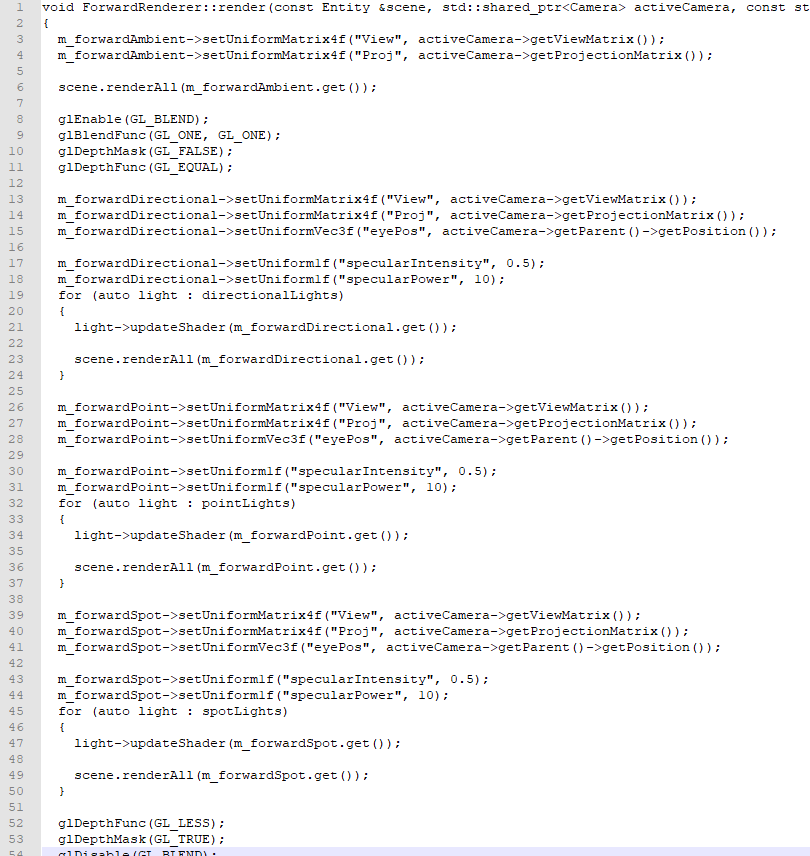
Below: The forward renderer drawing simple meshes



Below: The GUI Manager displaying diagnostics



Below: The forward rendering process



Below: The GUI manager rendering code. It contains some android-specific code by using the GLES2 define

void GuiManager::renderDrawLists(ImDrawData \*draw\_data)

{

// Avoid rendering when minimized, scale coordinates for retina displays (screen coordinates != framebuffer coordinates)

ImGuiIO &io = ImGui::GetIO();

int fb\_width = (int)(io.DisplaySize.x \* io.DisplayFramebufferScale.x);

int fb\_height = (int)(io.DisplaySize.y \* io.DisplayFramebufferScale.y);

if (fb\_width == 0 || fb\_height == 0)

return;

draw\_data->ScaleClipRects(io.DisplayFramebufferScale);

// Backup GL state

GLint last\_program;

glGetIntegerv(GL\_CURRENT\_PROGRAM, &last\_program);

GLint last\_texture;

glGetIntegerv(GL\_TEXTURE\_BINDING\_2D, &last\_texture);

GLint last\_active\_texture;

glGetIntegerv(GL\_ACTIVE\_TEXTURE, &last\_active\_texture);

GLint last\_array\_buffer;

glGetIntegerv(GL\_ARRAY\_BUFFER\_BINDING, &last\_array\_buffer);

GLint last\_element\_array\_buffer;

glGetIntegerv(GL\_ELEMENT\_ARRAY\_BUFFER\_BINDING, &last\_element\_array\_buffer);

#if !defined(GLES2)

GLint last\_vertex\_array;

glGetIntegerv(GL\_VERTEX\_ARRAY\_BINDING, &last\_vertex\_array);

GLint last\_blend\_src;

glGetIntegerv(GL\_BLEND\_SRC, &last\_blend\_src);

GLint last\_blend\_dst;

glGetIntegerv(GL\_BLEND\_DST, &last\_blend\_dst);

#endif

GLint last\_blend\_equation\_rgb;

glGetIntegerv(GL\_BLEND\_EQUATION\_RGB, &last\_blend\_equation\_rgb);

GLint last\_blend\_equation\_alpha;

glGetIntegerv(GL\_BLEND\_EQUATION\_ALPHA, &last\_blend\_equation\_alpha);

GLint last\_viewport[4];

glGetIntegerv(GL\_VIEWPORT, last\_viewport);

GLboolean last\_enable\_blend = glIsEnabled(GL\_BLEND);

GLboolean last\_enable\_cull\_face = glIsEnabled(GL\_CULL\_FACE);

GLboolean last\_enable\_depth\_test = glIsEnabled(GL\_DEPTH\_TEST);

GLboolean last\_enable\_scissor\_test = glIsEnabled(GL\_SCISSOR\_TEST);

// Setup render state: alpha-blending enabled, no face culling, no depth testing, scissor enabled

glEnable(GL\_BLEND);

glBlendEquation(GL\_FUNC\_ADD);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glDisable(GL\_CULL\_FACE);

glDisable(GL\_DEPTH\_TEST);

glEnable(GL\_SCISSOR\_TEST);

glActiveTexture(GL\_TEXTURE0);

// Setup orthographic projection matrix

glViewport(0, 0, (GLsizei)fb\_width, (GLsizei)fb\_height);

auto ortho\_projection = glm::ortho(0.0f, io.DisplaySize.x, io.DisplaySize.y, 0.0f);

m\_shader->bind();

m\_shader->setUniform1i("Texture", 0);

m\_shader->setUniformMatrix4f("ProjMtx", ortho\_projection);

#if !defined(GLES2)

glBindVertexArray(g\_VaoHandle);

#else

glBindBuffer(GL\_ARRAY\_BUFFER, g\_VboHandle);

glEnableVertexAttribArray(g\_AttribLocationPosition);

glEnableVertexAttribArray(g\_AttribLocationUV);

glEnableVertexAttribArray(g\_AttribLocationColor);

#define OFFSETOF(TYPE, ELEMENT) ((size\_t) & (((TYPE \*)0)->ELEMENT))

glVertexAttribPointer(g\_AttribLocationPosition, 2, GL\_FLOAT, GL\_FALSE, sizeof(ImDrawVert), (GLvoid \*)OFFSETOF(ImDrawVert, pos));

glVertexAttribPointer(g\_AttribLocationUV, 2, GL\_FLOAT, GL\_FALSE, sizeof(ImDrawVert), (GLvoid \*)OFFSETOF(ImDrawVert, uv));

glVertexAttribPointer(g\_AttribLocationColor, 4, GL\_UNSIGNED\_BYTE, GL\_TRUE, sizeof(ImDrawVert), (GLvoid \*)OFFSETOF(ImDrawVert, col));

#undef OFFSETOF

#endif

for (int n = 0; n < draw\_data->CmdListsCount; n++)

{

const ImDrawList \*cmd\_list = draw\_data->CmdLists[n];

const ImDrawIdx \*idx\_buffer\_offset = 0;

glBindBuffer(GL\_ARRAY\_BUFFER, g\_VboHandle);

glBufferData(GL\_ARRAY\_BUFFER, (GLsizeiptr)cmd\_list->VtxBuffer.size() \* sizeof(ImDrawVert), (GLvoid \*)&cmd\_list->VtxBuffer.front(), GL\_STREAM\_DRAW);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, g\_ElementsHandle);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, (GLsizeiptr)cmd\_list->IdxBuffer.size() \* sizeof(ImDrawIdx), (GLvoid \*)&cmd\_list->IdxBuffer.front(), GL\_STREAM\_DRAW);

for (const ImDrawCmd \*pcmd = cmd\_list->CmdBuffer.begin(); pcmd != cmd\_list->CmdBuffer.end(); pcmd++)

{

if (pcmd->UserCallback)

{

pcmd->UserCallback(cmd\_list, pcmd);

}

else

{

m\_textureData->bind(0);

glScissor((int)pcmd->ClipRect.x, (int)(fb\_height - pcmd->ClipRect.w), (int)(pcmd->ClipRect.z - pcmd->ClipRect.x), (int)(pcmd->ClipRect.w - pcmd->ClipRect.y));

glDrawElements(GL\_TRIANGLES, (GLsizei)pcmd->ElemCount, sizeof(ImDrawIdx) == 2 ? GL\_UNSIGNED\_SHORT : GL\_UNSIGNED\_INT, idx\_buffer\_offset);

}

idx\_buffer\_offset += pcmd->ElemCount;

}

}

#if defined(GLES2)

glDisableVertexAttribArray(g\_AttribLocationPosition);

glDisableVertexAttribArray(g\_AttribLocationUV);

glDisableVertexAttribArray(g\_AttribLocationColor);

#endif

// Restore modified GL state

glUseProgram(last\_program);

glActiveTexture(last\_active\_texture);

glBindTexture(GL\_TEXTURE\_2D, last\_texture);

#if !defined(GLES2)

glBindVertexArray(last\_vertex\_array);

#endif

glBindBuffer(GL\_ARRAY\_BUFFER, last\_array\_buffer);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, last\_element\_array\_buffer);

glBlendEquationSeparate(last\_blend\_equation\_rgb, last\_blend\_equation\_alpha);

#if !defined(GLES2)

glBlendFunc(last\_blend\_src, last\_blend\_dst);

#endif

if (last\_enable\_blend)

glEnable(GL\_BLEND);

else

glDisable(GL\_BLEND);

if (last\_enable\_cull\_face)

glEnable(GL\_CULL\_FACE);

else

glDisable(GL\_CULL\_FACE);

if (last\_enable\_depth\_test)

glEnable(GL\_DEPTH\_TEST);

else

glDisable(GL\_DEPTH\_TEST);

if (last\_enable\_scissor\_test)

glEnable(GL\_SCISSOR\_TEST);

else

glDisable(GL\_SCISSOR\_TEST);

glViewport(last\_viewport[0], last\_viewport[1], (GLsizei)last\_viewport[2], (GLsizei)last\_viewport[3]);

}