**LAMPIRAN**

**Lampiran 1**

// shader.vs

#version 330 core

layout (location = 0) in vec4 aPos;

uniform mat4 mvp;

void main()

{

gl\_Position = mvp \* aPos;

};

**Lampiran 2**

// shader.fs

#version 330 core

out vec4 FragColor;

uniform vec3 ourColor;

void main()

{

FragColor = vec4(ourColor, 1.0);

};

**Lampiran 3**

// floor.vs

#version 330 core

layout (location = 0) in vec3 aPos;

uniform mat4 mvp;

void main()

{

gl\_Position = mvp \* vec4(aPos, 1.0);

}

**Lampiran 4**

// floor.fs

#version 330 core

out vec4 FragColor;

uniform vec3 ourColor;

void main()

{

FragColor = vec4(ourColor, 1.0);

};

**Lampiran 5**

// bvh2.cpp

#include "bvh2.h"

#include <algorithm>

#include <cctype>

#include <functional>

#include <fstream>

#include <iostream>

// trim from start

static inline std::string &ltrim(std::string &s)

{

s.erase(s.begin(), std::find\_if(s.begin(), s.end(), std::not1(std::ptr\_fun<int, int>(std::isspace))));

return s;

}

// trim from end

static inline std::string &rtrim(std::string &s)

{

s.erase(std::find\_if(s.rbegin(), s.rend(), std::not1(std::ptr\_fun<int, int>(std::isspace))).base(), s.end());

return s;

}

// trim from both ends

static inline std::string &trim(std::string &s)

{

return ltrim(rtrim(s));

}

void deleteJoint(Joint\* joint)

{

if (joint == nullptr)

{

return;

}

for (Joint\* child : joint->children)

{

deleteJoint(child);

}

if (joint->channelsOrder != nullptr)

{

delete joint->channelsOrder;

}

delete joint;

}

void moveJoint(Joint\* joint, Motion\* motionData, int frameStartsIndex)

{

int startIndex = frameStartsIndex + joint->channelStart;

joint->matrix = glm::translate(glm::mat4(1.0f),

glm::vec3(joint->offset.x,

joint->offset.y,

joint->offset.z));

for (unsigned int i = 0; i < joint->numChannels; i++)

{

const short& channel = joint->channelsOrder[i];

float value = motionData->data[startIndex + i];

if (channel & Xposition)

joint->matrix = glm::translate(joint->matrix, glm::vec3(value, 0, 0));

if (channel & Yposition)

joint->matrix = glm::translate(joint->matrix, glm::vec3(0, value, 0));

if (channel & Zposition)

joint->matrix = glm::translate(joint->matrix, glm::vec3(0, 0, value));

if (channel & Xrotation)

joint->matrix = glm::rotate(joint->matrix, glm::radians(value), glm::vec3(1, 0, 0));

if (channel & Yrotation)

joint->matrix = glm::rotate(joint->matrix, glm::radians(value), glm::vec3(0, 1, 0));

if (channel & Zrotation)

joint->matrix = glm::rotate(joint->matrix, glm::radians(value), glm::vec3(0, 0, 1));

}

if (joint->parent != nullptr)

joint->matrix = joint->parent->matrix \* joint->matrix;

for (auto& child : joint->children)

moveJoint(child, motionData, frameStartsIndex);

}

Bvh2::Bvh2()

:

rootJoint(nullptr),

jointNames()

{

motionData.data = 0;

}

Bvh2::~Bvh2()

{

jointNames.clear();

deleteJoint(rootJoint);

if (motionData.data != nullptr)

{

delete[] motionData.data;

}

}

void Bvh2::printJoint(const Joint \* const joint) const

{

std::cout << "joint: \t" << joint->name << std::endl;

for (std::vector<Joint\*>::const\_iterator ct = joint->children.begin();

ct != joint->children.end();

++ct)

{

Joint\* \_tmp = \*ct;

if (\_tmp->children.size() > 0)

{

printJoint(\_tmp);

}

}

}

void Bvh2::load(const std::string & filename)

{

std::fstream file;

file.open(filename.c\_str(), std::ios\_base::in);

if (file.is\_open())

{

std::string line;

while (file.good())

{

file >> line;

if (trim(line) == "HIERARCHY")

{

loadHierarchy(file);

}

break;

}

file.close();

}

setJointNames(rootJoint);

}

void Bvh2::testOutput() const

{

if (rootJoint == nullptr)

return;

std::cout << "num frames: " << motionData.numFrames << std::endl;

std::cout << "num motion channels: " << motionData.numMotionChannels << std::endl;

}

void Bvh2::moveTo(unsigned int frame)

{

unsigned int startIndex = frame \* motionData.numMotionChannels;

moveJoint(rootJoint, &motionData, startIndex);

}

void Bvh2::setJointNames(const Joint\* const joint)

{

//jointNames.push\_back(joint->name);

//for (std::vector<Joint\*>::const\_iterator ct = joint->children.begin();

// ct != joint->children.end();

// ++ct)

//{

// Joint\* \_tmp = \*ct;

// if (\_tmp->children.size() > 0)

// {

// setJointNames(\_tmp);

// }

//}

jointNames.push\_back(joint->name);

for (std::vector<Joint\*>::const\_iterator ct = joint->children.begin();

ct != joint->children.end();

++ct)

{

Joint\* \_tmp = \*ct;

setJointNames(\_tmp);

}

}

Joint \* Bvh2::loadJoint(std::istream & stream, Joint \* parent)

{

Joint\* joint = new Joint;

joint->parent = parent;

joint->matrix = glm::mat4(1.0f);

std::string\* name = new std::string;

stream >> \*name;

joint->name = name->c\_str();

std::string tmp;

joint->matrix = glm::mat4(1.0f);

static int \_channelStart = 0;

unsigned channelOrderIndex = 0;

while (stream.good())

{

stream >> tmp;

tmp = trim(tmp);

char c = tmp.at(0);

if (c == 'X' || c == 'Y' || c == 'Z')

{

if (tmp == "Xposition")

joint->channelsOrder[channelOrderIndex++] = Xposition;

if (tmp == "Yposition")

joint->channelsOrder[channelOrderIndex++] = Yposition;

if (tmp == "Zposition")

joint->channelsOrder[channelOrderIndex++] = Zposition;

if (tmp == "Xrotation")

joint->channelsOrder[channelOrderIndex++] = Xrotation;

if (tmp == "Yrotation")

joint->channelsOrder[channelOrderIndex++] = Yrotation;

if (tmp == "Zrotation")

joint->channelsOrder[channelOrderIndex++] = Zrotation;

}

if (tmp == "OFFSET")

{

stream >> joint->offset.x >> joint->offset.y >> joint->offset.z;

glm::mat4 mat = joint->parent == nullptr ? glm::mat4(1.0f) : joint->parent->matrix;

}

else if (tmp == "CHANNELS")

{

stream >> joint->numChannels;

motionData.numMotionChannels += joint->numChannels;

joint->channelStart = \_channelStart;

\_channelStart += joint->numChannels;

joint->channelsOrder = new short[joint->numChannels];

}

else if (tmp == "JOINT")

{

Joint\* tmpJoint = loadJoint(stream, joint);

tmpJoint->parent = joint;

joint->children.push\_back(tmpJoint);

}

else if (tmp == "End")

{

stream >> tmp >> tmp;

Joint\* tmpJoint = new Joint;

tmpJoint->parent = joint;

tmpJoint->numChannels = 0;

tmpJoint->name = "EndSite";

joint->children.push\_back(tmpJoint);

stream >> tmp;

if (tmp == "OFFSET")

stream >> tmpJoint->offset.x >> tmpJoint->offset.y >> tmpJoint->offset.z;

stream >> tmp;

}

else if (tmp == "}")

{

return joint;

}

}

return joint;

}

void Bvh2::loadHierarchy(std::istream & stream)

{

std::string tmp;

while (stream.good())

{

stream >> tmp;

if (trim(tmp) == "ROOT")

rootJoint = loadJoint(stream);

else if (trim(tmp) == "MOTION")

loadMotion(stream);

}

}

void Bvh2::loadMotion(std::istream & stream)

{

std::string tmp;

while (stream.good())

{

stream >> tmp;

if (trim(tmp) == "Frames:")

{

stream >> motionData.numFrames;

}

else if (trim(tmp) == "Frame")

{

float frameTime;

stream >> tmp >> frameTime;

int numFrames = motionData.numFrames;

int numChannels = motionData.numMotionChannels;

motionData.data = new float[numFrames \* numChannels];

for (int frame = 0; frame < numFrames; frame++)

{

for (int channel = 0; channel < numChannels; channel++)

{

float x;

std::stringstream ss;

stream >> tmp;

ss << tmp;

ss >> x;

int index = frame \* numChannels + channel;

motionData.data[index] = x;

}

}

}

}

}

**Lampiran 6**

// main.cpp

int main(int argc, char\* argv[])

{

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

GLFWwindow\* window = glfwCreateWindow(screenWidth, screenHeight, "PENULISAN ILMIAH", nullptr, nullptr);

if (window == nullptr)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, frameBufferSizeCallback);

glfwSetCursorPosCallback(window, mouseCallback);

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize Glad" << std::endl;

return -1;

}

glEnable(GL\_DEPTH\_TEST);

glfwSwapInterval(0);

// floor

float floorVertices[] = {

100.0f, 0.0f, 100.0f,

100.0f, 0.0f, -100.0f,

-100.0f, 0.0f, -100.0f,

-100.0f, 0.0f, 100.0f

};

unsigned int floorIndices[] = {

0, 1, 3,

1, 2, 3

};

deltaTime = currentTime - lastFrame;

lastFrame = currentTime;

glLineWidth(boneWidth);

glPointSize(jointPointSize);

// pre render calculation

view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

projection = glm::perspective(glm::radians(fov), (float)screenWidth / (float)screenHeight,

0.1f, 1000.0f);

mvp = projection \* view \* model;

// draw floor

floorShader.use();

floorShader.setMat4("mvp", mvp);

floorShader.setVec3("ourColor", floorColor[0], floorColor[1], floorColor[2]);

glBindVertexArray(floorVAO);

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0);

// skeleton

bvhShader.use();

bvhShader.setMat4("mvp", mvp);

updateBvh();

if (renderBones)

{

glBindVertexArray(bvhVAO);

bvhShader.setVec3("ourColor", boneColor[0], boneColor[1], boneColor[2]);

glDrawElements(GL\_LINES, bvhElements, GL\_UNSIGNED\_SHORT, (void\*)0);

}

if (renderJoints)

{

glBindVertexArray(bvhVAO);

bvhShader.setVec3("ourColor", jointColor[0], jointColor[1], jointColor[2]);

glDrawElements(GL\_POINTS, bvhElements, GL\_UNSIGNED\_SHORT, (void\*)0);

}

// com

processCOM(bvhVertices, comVertices);

if (renderBodyCOM)

{

floorShader.setVec3("ourColor", comColor[0], comColor[1], comColor[2]);

glBindVertexArray(comVAO);

glDrawArrays(GL\_POINTS, 0, (int)comVertices.size());

}

if (renderSegmentCOM)

{

floorShader.setVec3("ourColor", segmentComColor[0], segmentComColor[1], segmentComColor[2]);

glBindVertexArray(segmentsCogVAO);

glDrawArrays(GL\_POINTS, 0, (int)segmentsCogVertices.size());

}

// BVH Player Settings;

{

ImGui::Begin("BVH Player Settings");

ImGui::SliderInt("Frame", &bvhFrame, 0, bvh->getNumFrames());

ImGui::SameLine();

ImGui::Checkbox("Loop", &loop);

ImGui::SameLine();

if (ImGui::Button("Play / Pause"))

frameChange = !frameChange;

ImGui::SameLine();

if (ImGui::Button("<") && bvhFrame != 0)

bvhFrame--;

ImGui::SameLine();

if (ImGui::Button(">") && bvhFrame != bvh->getNumFrames())

bvhFrame++;

ImGui::Checkbox("Render Bones", &renderBones);

ImGui::SameLine();

ImGui::Checkbox("Render Joints", &renderJoints);

ImGui::SameLine();

ImGui::Checkbox("Render Segments COM", &renderSegmentCOM);

ImGui::SameLine();

ImGui::Checkbox("Render Body COM", &renderBodyCOM);

//ImGui::SameLine();

//ImGui::InputInt("Desired FPS", &FPS);

ImGui::Text("%.1f FPS", ImGui::GetIO().Framerate);

ImGui::Text("Application average %.3f ms/frame", 1000.0f / ImGui::GetIO().Framerate);

ImGui::Text("Number of frames: %i", bvh->getNumFrames());

//if (ImGui::CollapsingHeader("Display Settings"))

{

ImGui::PushItemWidth(200);

ImGui::ColorEdit3("Floor Color", floorColor);

ImGui::SameLine();

ImGui::ColorEdit3("Background Color", backgroundColor);

ImGui::SameLine();

ImGui::ColorEdit3("Bone Color ", boneColor);

ImGui::SameLine();

ImGui::ColorEdit3("Joint Color", jointColor);

ImGui::SameLine();

ImGui::ColorEdit3("COM Color", comColor);

ImGui::SliderFloat("Bone Width ", &boneWidth, 0.001f, 10.0f);

ImGui::SameLine();

ImGui::SliderFloat("Joint Size ", &jointPointSize, 0.001f, 10.0f);

ImGui::SameLine();

ImGui::ColorEdit3("Segments COM Color", segmentComColor);

ImGui::PopItemWidth();

}

ImGui::End();

}

// BVH Stats

{

ImGui::Begin("BVH Status");

auto nameVector = bvh->getJointNames();

for (size\_t i = 0; i < nameVector.size(); i++)

{

if (nameVector[i] == "EndSite")

nameVector[i] = nameVector[i - 1] + nameVector[i];

}

if (ImGui::CollapsingHeader("Joints' World X Y Z Positions"))

{

for (size\_t i = 0; i < nameVector.size(); i++)

{

glm::vec3 channels = glm::vec3(bvhVertices[i].x, bvhVertices[i].y, bvhVertices[i].z);

nameVector[i].append(" [");

nameVector[i].append(std::to\_string(i));

nameVector[i].append("]");

ImGui::InputFloat3(nameVector[i].c\_str(), &channels[0], "%.6f", ImGuiInputTextFlags\_ReadOnly);

}

}

if (ImGui::CollapsingHeader("COM Properties"))

{

ImGui::Text(" ");

ImGui::Separator();

ImGui::Columns(1);

ImGui::Text("Gender");

ImGui::Separator();

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::RadioButton("Male", &selectedGender, 0);

ImGui::NextColumn();

ImGui::RadioButton("Female", &selectedGender, 1);

ImGui::Columns(1);

ImGui::InputFloat("Total Body Weight", &totalBodyWeight);

ImGui::Separator();

ImGui::Text(" ");

ImGui::Columns(1);

ImGui::Separator();

ImGui::Text("Segment Mass Percent");

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Head & Neck Mass Male", &headNeckMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Head & Neck Mass Female", &headNeckMassPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Trunk Mass Male", &trunkMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Trunk Mass Female", &trunkMassPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Upper Arm Mass Male", &upperArmMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Upper Arm Mass Female", &upperArmMassPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Fore Arm Mass Male", &foreArmMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Fore Arm Mass Female", &foreArmMassPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Hand Mass Male", &handMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Hand Mass Female", &handMassPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Thigh Mass Male", &thighMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Thigh Mass Female", &thighMassPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Shank Mass Male", &shankMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Shank Mass Female", &shankMassPercent[1]);

ImGui::Columns(1);

graphFrames, 0, "", -leftForeArmGraphZHeight, leftForeArmGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Fore Arm COM Z Height", &leftForeArmGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Right Fore Arm COM"))

{

rightForeArmGraph[0][bvhFrame] = segmentsCogVertices[5].x;

rightForeArmGraph[1][bvhFrame] = segmentsCogVertices[5].y;

rightForeArmGraph[2][bvhFrame] = segmentsCogVertices[5].z;

static float rightForeArmGraphXHeight = 150.0f;

static float rightForeArmGraphYHeight = 150.0f;

static float rightForeArmGraphZHeight = 150.0f;

ImGui::PlotHistogram("Right Fore Arm COM X", &rightForeArmGraph[0][0], graphFrames, 0, "", -rightForeArmGraphXHeight, rightForeArmGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Fore Arm COM X Height", &rightForeArmGraphXHeight, 1, 200);

ImGui::PlotHistogram("Right Fore Arm COM Y", &rightForeArmGraph[1][0], graphFrames, 0, "", -rightForeArmGraphYHeight, rightForeArmGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Fore Arm COM Y Height", &rightForeArmGraphYHeight, 1, 200);

ImGui::PlotHistogram("Right Fore Arm COM Z", &rightForeArmGraph[2][0], graphFrames, 0, "", -rightForeArmGraphZHeight, rightForeArmGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Fore Arm COM Z Height", &rightForeArmGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Left Hand COM"))

{

leftHandGraph[0][bvhFrame] = segmentsCogVertices[6].x;

leftHandGraph[1][bvhFrame] = segmentsCogVertices[6].y;

leftHandGraph[2][bvhFrame] = segmentsCogVertices[6].z;

static float leftHandGraphXHeight = 150.0f;

static float leftHandGraphYHeight = 150.0f;

static float leftHandGraphZHeight = 150.0f;

ImGui::PlotHistogram("Left Hand COM X", &leftHandGraph[0][0], graphFrames, 0, "", -leftHandGraphXHeight, leftHandGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Hand COM X Height", &leftHandGraphXHeight, 1, 200);

ImGui::PlotHistogram("Left Hand COM Y", &leftHandGraph[1][0], graphFrames, 0, "", -leftHandGraphYHeight, leftHandGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Hand COM Y Height", &leftHandGraphYHeight, 1, 200);

ImGui::PlotHistogram("Left Hand COM Z", &leftHandGraph[2][0], graphFrames, 0, "", -leftHandGraphZHeight, leftHandGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Hand COM Z Height", &leftHandGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Right Hand COM"))

{

rightHandGraph[0][bvhFrame] = segmentsCogVertices[7].x;

rightHandGraph[1][bvhFrame] = segmentsCogVertices[7].y;

rightHandGraph[2][bvhFrame] = segmentsCogVertices[7].z;

static float rightHandGraphXHeight = 150.0f;

static float rightHandGraphYHeight = 150.0f;

static float rightHandGraphZHeight = 150.0f;

ImGui::PlotHistogram("Right Hand COM X", &rightHandGraph[0][0], graphFrames, 0, "", -rightHandGraphXHeight, rightHandGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Hand COM X Height", &rightHandGraphXHeight, 1, 200);

ImGui::PlotHistogram("Right Hand COM Y", &rightHandGraph[1][0], graphFrames, 0, "", -rightHandGraphYHeight, rightHandGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Hand COM Y Height", &rightHandGraphYHeight, 1, 200);

ImGui::PlotHistogram("Right Hand COM Z", &rightHandGraph[2][0], graphFrames, 0, "", -rightHandGraphZHeight, rightHandGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Hand COM Z Height", &rightHandGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Left Thigh COM"))

{

leftThighGraph[0][bvhFrame] = segmentsCogVertices[8].x;

leftThighGraph[1][bvhFrame] = segmentsCogVertices[8].y;

leftThighGraph[2][bvhFrame] = segmentsCogVertices[8].z;

static float leftThighGraphXHeight = 150.0f;

static float leftThighGraphYHeight = 150.0f;

static float leftThighGraphZHeight = 150.0f;

ImGui::PlotHistogram("Left Thigh COM X", &leftThighGraph[0][0], graphFrames, 0, "", -leftThighGraphXHeight, leftThighGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Thigh COM X Height", &leftThighGraphXHeight, 1, 200);

ImGui::PlotHistogram("Left Thigh COM Y", &leftThighGraph[1][0], graphFrames, 0, "", -leftThighGraphYHeight, leftThighGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Thigh COM Y Height", &leftThighGraphYHeight, 1, 200);

ImGui::PlotHistogram("Left Thigh COM Z", &leftThighGraph[2][0], graphFrames, 0, "", -leftThighGraphZHeight, leftThighGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Thigh COM Z Height", &leftThighGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Right Thigh COM"))

{

rightThighGraph[0][bvhFrame] = segmentsCogVertices[9].x;

rightThighGraph[1][bvhFrame] = segmentsCogVertices[9].y;

rightThighGraph[2][bvhFrame] = segmentsCogVertices[9].z;

static float rightThighGraphXHeight = 150.0f;

static float rightThighGraphYHeight = 150.0f;

static float rightThighGraphZHeight = 150.0f;

ImGui::PlotHistogram("Right Thigh COM X", &rightThighGraph[0][0], graphFrames, 0, "", -rightThighGraphXHeight, rightThighGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Thigh COM X Height", &rightThighGraphXHeight, 1, 200);

ImGui::PlotHistogram("Right Thigh COM Y", &rightThighGraph[1][0], graphFrames, 0, "", -rightThighGraphYHeight, rightThighGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Thigh COM Y Height", &rightThighGraphYHeight, 1, 200);

ImGui::PlotHistogram("Right Thigh COM Z", &rightThighGraph[2][0], graphFrames, 0, "", -rightThighGraphZHeight, rightThighGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Thigh COM Z Height", &rightThighGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Left Shank COM"))

{

leftShankGraph[0][bvhFrame] = segmentsCogVertices[10].x;

leftShankGraph[1][bvhFrame] = segmentsCogVertices[10].y;

leftShankGraph[2][bvhFrame] = segmentsCogVertices[10].z;

static float leftShankGraphXHeight = 150.0f;

static float leftShankGraphYHeight = 150.0f;

static float leftShankGraphZHeight = 150.0f;

ImGui::PlotHistogram("Left Shank COM X", &leftShankGraph[0][0], graphFrames, 0, "", -leftShankGraphXHeight, leftShankGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Shank COM X Height", &leftShankGraphXHeight, 1, 200);

ImGui::PlotHistogram("Left Shank COM Y", &leftShankGraph[1][0], graphFrames, 0, "", -leftShankGraphYHeight, leftShankGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Shank COM Y Height", &leftShankGraphYHeight, 1, 200);

ImGui::PlotHistogram("Left Shank COM Z", &leftShankGraph[2][0], graphFrames, 0, "", -leftShankGraphZHeight, leftShankGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Shank COM Z Height", &leftShankGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Right Shank COM"))

{

rightShankGraph[0][bvhFrame] = segmentsCogVertices[11].x;

rightShankGraph[1][bvhFrame] = segmentsCogVertices[11].y;

rightShankGraph[2][bvhFrame] = segmentsCogVertices[11].z;

ImGui::SliderFloat("Left Foot COM X Height", &leftFootGraphXHeight, 1, 200);

ImGui::PlotHistogram("Left Foot COM Y", &leftFootGraph[1][0], graphFrames, 0, "", -leftFootGraphYHeight, leftFootGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Foot COM Y Height", &leftFootGraphYHeight, 1, 200);

ImGui::PlotHistogram("Left Foot COM Z", &leftFootGraph[2][0], graphFrames, 0, "", -leftFootGraphZHeight, leftFootGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Foot COM Z Height", &leftFootGraphZHeight, 1, 200);

}

ImGui::Render();

ImGui\_ImplOpenGL3\_RenderDrawData(ImGui::GetDrawData());

glfwSwapBuffers(window);

while (glfwGetTime() < lastTimeFrame + 1.0 / FPS)

{

Sleep(10);

}

lastTimeFrame += 1.0 / FPS;

}

glfwTerminate();

return 0;

}