**LAMPIRAN**

**Lampiran 1**

// Shader.h

#pragma once

#include <glad/glad.h>

#include <glm/glm.hpp>

#include <string>

#include <fstream>

#include <sstream>

#include <iostream>

class Shader

{

public:

unsigned int ID;

Shader(const char\* vertexPath, const char\* fragmentPath);

void use();

void setBool(const std::string& name, bool value) const;

void setInt(const std::string& name, int value) const;

void setFloat(const std::string& name, float value) const;

void setVec2(const std::string& name, const glm::vec2& value) const;

void setVec2(const std::string& name, float x, float y) const;

void setVec3(const std::string& name, const glm::vec3& value) const;

void setVec3(const std::string& name, float x, float y, float z) const;

void setVec4(const std::string& name, const glm::vec4& value) const;

void setVec4(const std::string& name, float x, float y, float z, float w) const;

void setMat2(const std::string& name, const glm::mat2& mat) const;

void setMat3(const std::string& name, const glm::mat3& mat) const;

void setMat4(const std::string& name, const glm::mat4& mat) const;

private:

void checkCompileErrors(unsigned int shader, std::string type);

};

**Lampiran 2**

// Shader.cpp

#include "Shader.h"

Shader::Shader(const char \* vertexPath, const char \* fragmentPath)

{

std::string vertexCode;

std::string fragmentCode;

std::ifstream vShaderFile;

std::ifstream fShaderFile;

vShaderFile.exceptions (std::ifstream::failbit | std::ifstream::badbit);

fShaderFile.exceptions (std::ifstream::failbit | std::ifstream::badbit);

try

{

vShaderFile.open(vertexPath);

fShaderFile.open(fragmentPath);

std::stringstream vShaderStream, fShaderStream;

// read file's buffer contents into streams

vShaderStream << vShaderFile.rdbuf();

fShaderStream << fShaderFile.rdbuf();

// close file handlers

vShaderFile.close();

fShaderFile.close();

// convert stream into string

vertexCode = vShaderStream.str();

fragmentCode = fShaderStream.str();

}

catch (std::ifstream::failure e)

{

std::cout << "ERROR::SHADER::FILE\_NOT\_SUCCESFULLY\_READ" << std::endl;

}

const char\* vShaderCode = vertexCode.c\_str();

const char \* fShaderCode = fragmentCode.c\_str();

// 2. compile shaders

unsigned int vertex, fragment;

// vertex shader

vertex = glCreateShader(GL\_VERTEX\_SHADER);

glShaderSource(vertex, 1, &vShaderCode, NULL);

glCompileShader(vertex);

checkCompileErrors(vertex, "VERTEX");

// fragment Shader

fragment = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(fragment, 1, &fShaderCode, NULL);

glCompileShader(fragment);

checkCompileErrors(fragment, "FRAGMENT");

// shader Program

ID = glCreateProgram();

glAttachShader(ID, vertex);

glAttachShader(ID, fragment);

glLinkProgram(ID);

checkCompileErrors(ID, "PROGRAM");

// delete the shaders as they're linked into our program now and no longer necessary

glDeleteShader(vertex);

glDeleteShader(fragment);

}

void Shader::use()

{

glUseProgram(ID);

}

void Shader::setBool(const std::string & name, bool value) const

{

glUniform1i(glGetUniformLocation(ID, name.c\_str()), (int)value);

}

void Shader::setInt(const std::string & name, int value) const

{

glUniform1i(glGetUniformLocation(ID, name.c\_str()), value);

}

void Shader::setFloat(const std::string & name, float value) const

{

glUniform1f(glGetUniformLocation(ID, name.c\_str()), value);

}

void Shader::setVec2(const std::string & name, const glm::vec2 & value) const

{

glUniform2fv(glGetUniformLocation(ID, name.c\_str()), 1, &value[0] );

}

void Shader::setVec2(const std::string & name, float x, float y) const

{

glUniform2f(glGetUniformLocation(ID, name.c\_str()), x, y);

}

void Shader::setVec3(const std::string & name, const glm::vec3 & value) const

{

glUniform3fv(glGetUniformLocation(ID, name.c\_str()), 1, &value[0] );

}

void Shader::setVec3(const std::string & name, float x, float y, float z) const

{

glUniform3f(glGetUniformLocation(ID, name.c\_str()), x, y, z);

}

void Shader::setVec4(const std::string & name, const glm::vec4 & value) const

{

glUniform4fv(glGetUniformLocation(ID, name.c\_str()), 1, &value[0] );

}

void Shader::setVec4(const std::string & name, float x, float y, float z, float w) const

{

glUniform4f(glGetUniformLocation(ID, name.c\_str()), x, y, z, w);

}

void Shader::setMat2(const std::string & name, const glm::mat2 & mat) const

{

glUniformMatrix2fv(glGetUniformLocation(ID, name.c\_str()), 1, GL\_FALSE, &mat[0][0]);

}

void Shader::setMat3(const std::string & name, const glm::mat3 & mat) const

{

glUniformMatrix3fv(glGetUniformLocation(ID, name.c\_str()), 1, GL\_FALSE, &mat[0][0]);

}

void Shader::setMat4(const std::string & name, const glm::mat4 & mat) const

{

glUniformMatrix4fv(glGetUniformLocation(ID, name.c\_str()), 1, GL\_FALSE, &mat[0][0]);

}

void Shader::checkCompileErrors(unsigned int shader, std::string type)

{

int success;

char infoLog[1024];

if (type != "PROGRAM")

{

glGetShaderiv(shader, GL\_COMPILE\_STATUS, &success);

if (!success)

{

glGetShaderInfoLog(shader, 1024, NULL, infoLog);

std::cout << "ERROR::SHADER\_COMPILATION\_ERROR of type: "

<< type << std::endl << infoLog << std::endl;

}

}

else

{

glGetProgramiv(shader, GL\_LINK\_STATUS, &success);

if (!success)

{

glGetProgramInfoLog(shader, 1024, NULL, infoLog);

std::cout << "ERROR::PROGRAM\_LINKING\_ERROR of type: "

<< type << std::endl << infoLog << std::endl;

}

}

}

**Lampiran 3**

// bvh2.h

#pragma once

#include <string>

#include <vector>

#include <sstream>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#define Xposition 0x01

#define Yposition 0x02

#define Zposition 0x04

#define Zrotation 0x10

#define Xrotation 0x20

#define Yrotation 0x40

struct Offset

{

float x, y, z;

};

struct Joint

{

const char\* name = nullptr;

Joint\* parent = nullptr;

Offset offset;

unsigned int numChannels = 0;

short\* channelsOrder = nullptr;

std::vector<Joint\*> children;

glm::mat4 matrix;

unsigned int channelStart = 0;

};

struct Hierarchy

{

Joint\* rootJoint;

int numChanneles;

};

struct Motion

{

unsigned int numFrames;

unsigned int numMotionChannels = 0;

float\* data = nullptr;

unsigned int\* jointChannelsOffsets;

};

class Bvh2

{

public:

Bvh2();

~Bvh2();

void printJoint(const Joint\* const joint) const;

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

void testOutput() const;

void moveTo(unsigned int frame);

const Joint\* getRootJoint() const { return rootJoint; }

unsigned int getNumFrames() const { return motionData.numFrames - 1; }

std::vector<std::string> getJointNames() { return jointNames; };

private:

Joint\* loadJoint(std::istream& stream, Joint\* parent = nullptr);

void loadHierarchy(std::istream& stream);

void loadMotion(std::istream& stream);

void setJointNames(const Joint\* const joint);

private:

Joint\* rootJoint;

Motion motionData;

std::vector<std::string> jointNames;

};

**Lampiran 4**

// 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 5**

// main.cpp

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <imgui.h>

#include <imgui\_impl\_glfw.h>

#include <imgui\_impl\_opengl3.h>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include <glm/gtc/matrix\_inverse.hpp>

#include <iostream>

#include "Shader.h"

#include "bvh2.h"

// GLFW callbacks declarations

void frameBufferSizeCallback(GLFWwindow\* window, int width, int height);

void processInput(GLFWwindow\* window);

void mouseCallback(GLFWwindow\* window, double xpos, double ypos);

// renderer settings

unsigned int screenWidth = 1800;

unsigned int screenHeight = 1000;

int FPS = 100;

float deltaTime = 0.0f;

float lastFrame = 0.0f;

bool loop = false;

bool renderBones = true;

bool renderJoints = true;

bool renderSegmentCOM = false;

bool renderBodyCOM = true;

// camera settings

glm::vec3 cameraPos = glm::vec3(100.0f, 70.0f, 300.0f);

glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

float cameraSpeed = 2.5f;

bool firstMouse = true;

float yaw = -90.0f;

float pitch = 0.0f;

float lastX = screenWidth / 2.0f;

float lastY = screenHeight / 2.0f;

float fov = 45.0f;

float backgroundColor[3] = { 0.2f, 0.2f, 0.2f };

float floorColor[3] = { 0.05f, 0.05f, 0.05f };

float boneColor[3] = { 0.5f, 0.5f, 0.5f };

float jointColor[3] = { 0.5f, 1.0f, 1.0f };

float segmentComColor[3] = { 1.0f, 1.0f, 0.0f };

float comColor[3] = { 1.0f, 0.0f, 0.0f };

// COM properties

int selectedGender = 0;

float totalBodyWeight = 60.0f;

float headNeckMassPercent[2] = { 6.94f, 6.68f };

float trunkMassPercent[2] = { 43.46f, 42.58f };

float upperArmMassPercent[2] = { 2.71f, 2.55f };

float foreArmMassPercent[2] = { 1.62f, 1.38f };

float handMassPercent[2] = { 0.61f, 0.56f };

float thighMassPercent[2] = { 14.16f, 14.78f };

float shankMassPercent[2] = { 4.33f, 4.81f };

float footMassPercent[2] = { 1.37f, 1.29f };

float headNeckLengthPercent[2] = { 50.02f, 48.41f };

float trunkLengthPercent[2] = { 43.10f, 37.82f };

float upperArmLengthPercent[2] = { 57.72f, 57.54f };

float foreArmLengthPercent[2] = { 45.74f, 45.59f };

float handLengthPercent[2] = { 79.00f, 74.74f };

float thighLengthPercent[2] = { 40.95f, 36.12f };

float shankLengthPercent[2] = { 43.95f, 43.52f };

float footLengthPercent[2] = { 44.15f, 40.14f };

// bvh settings

float boneWidth = 1.0;

float jointPointSize = 2.0;

Bvh2\* bvh;

unsigned int bvhVBO, bvhEBO, bvhVAO;

std::vector<glm::vec4> bvhVertices;

std::vector<short> bvhIndices;

short bvhElements = 0;

int bvhFrame = 0;

bool frameChange = false;

unsigned int comVBO, comVAO;

std::vector<glm::vec4> comVertices;

unsigned int segmentsCogVBO, segmentsCogVAO;

std::vector<glm::vec4> segmentsCogVertices;

/\*################################################################################################################################################\*/

void processBvh(Joint\* joint, std::vector<glm::vec4>& vertices,

std::vector<short>& indices, short parentIndex = 0)

{

glm::vec4 translatedVertex = joint->matrix[3];

vertices.push\_back(translatedVertex);

short myindex = (short)(vertices.size() - 1);

if (parentIndex != myindex)

{

indices.push\_back(parentIndex);

indices.push\_back(myindex);

}

for (auto& child : joint->children)

{

processBvh(child, vertices, indices, myindex);

}

}

glm::vec4 myLerp(glm::vec4 x, glm::vec4 y, float t) {

return x \* (1.f - t / 100.0f) + y \* (t / 100.0f);

}

void processCOM(const std::vector<glm::vec4>& bvhVertices, std::vector<glm::vec4>& comVertices)

{

comVertices.clear();

segmentsCogVertices.clear();

comVertices.clear();

glm::vec4 headNeck = myLerp(bvhVertices[3], bvhVertices[5], headNeckLengthPercent[selectedGender]);

glm::vec4 trunk = myLerp(bvhVertices[0], bvhVertices[2], trunkLengthPercent[selectedGender]);

glm::vec4 leftUpperArm = myLerp(bvhVertices[6], bvhVertices[8], upperArmLengthPercent[selectedGender]);

glm::vec4 rightUpperArm = myLerp(bvhVertices[11], bvhVertices[13], upperArmLengthPercent[selectedGender]);

glm::vec4 leftForeArm = myLerp(bvhVertices[8], bvhVertices[9], foreArmLengthPercent[selectedGender]);

glm::vec4 rightForeArm = myLerp(bvhVertices[13], bvhVertices[14], foreArmLengthPercent[selectedGender]);

// TODO:(denilson) WRIST JOINT to MCP3(the middle finger base joint!), make some interpolation

glm::vec4 leftHand = myLerp(bvhVertices[9], bvhVertices[10], handLengthPercent[selectedGender]);

glm::vec4 rightHand = myLerp(bvhVertices[14], bvhVertices[15], handLengthPercent[selectedGender]);

glm::vec4 leftThigh = myLerp(bvhVertices[16], bvhVertices[17], thighLengthPercent[selectedGender]);

glm::vec4 rightThigh = myLerp(bvhVertices[21], bvhVertices[22], thighLengthPercent[selectedGender]);

glm::vec4 leftShank = myLerp(bvhVertices[17], bvhVertices[18], shankLengthPercent[selectedGender]);

glm::vec4 rightShank = myLerp(bvhVertices[22], bvhVertices[23], shankLengthPercent[selectedGender]);

glm::vec4 leftFoot = myLerp(bvhVertices[18], bvhVertices[20], footLengthPercent[selectedGender]);

glm::vec4 rightFoot = myLerp(bvhVertices[23], bvhVertices[25], footLengthPercent[selectedGender]);

// body COM

float headNeckMass = (headNeckMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float trunkMass = (trunkMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float upperArmMass = (upperArmMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float foreArmMass = (foreArmMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float handMass = (handMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float thighMass = (thighMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float shankMass = (shankMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float footMass = (footMassPercent[selectedGender] / 100.0f) \* totalBodyWeight;

float bodyCOMX = (

(headNeck.x \* headNeckMass) +

(trunk.x \* trunkMass) +

(leftUpperArm.x \* upperArmMass) +

(rightUpperArm.x \* upperArmMass) +

(leftForeArm.x \* foreArmMass) +

(rightForeArm.x \* foreArmMass) +

(leftHand.x \* handMass) +

(rightHand.x \* handMass) +

(leftThigh.x \* thighMass) +

(rightThigh.x \* thighMass) +

(leftShank.x \* shankMass) +

(rightShank.x \* shankMass) +

(leftFoot.x \* footMass) +

(rightFoot.x \* footMass)

) / totalBodyWeight;

float bodyCOMY = (

(headNeck.y \* headNeckMass) +

(trunk.y \* trunkMass) +

(leftUpperArm.y \* upperArmMass) +

(rightUpperArm.y \* upperArmMass) +

(leftForeArm.y \* foreArmMass) +

(rightForeArm.y \* foreArmMass) +

(leftHand.y \* handMass) +

(rightHand.y \* handMass) +

(leftThigh.y \* thighMass) +

(rightThigh.y \* thighMass) +

(leftShank.y \* shankMass) +

(rightShank.y \* shankMass) +

(leftFoot.y \* footMass) +

(rightFoot.y \* footMass)

) / totalBodyWeight;

float bodyCOMZ = (

(headNeck.z \* headNeckMass) +

(trunk.z \* trunkMass) +

(leftUpperArm.z \* upperArmMass) +

(rightUpperArm.z \* upperArmMass) +

(leftForeArm.z \* foreArmMass) +

(rightForeArm.z \* foreArmMass) +

(leftHand.z \* handMass) +

(rightHand.z \* handMass) +

(leftThigh.z \* thighMass) +

(rightThigh.z \* thighMass) +

(leftShank.z \* shankMass) +

(rightShank.z \* shankMass) +

(leftFoot.z \* footMass) +

(rightFoot.z \* footMass)

) / totalBodyWeight;

glm::vec4 bodyCOM = glm::vec4(bodyCOMX, bodyCOMY, bodyCOMZ, 1.0f);

// push

segmentsCogVertices.push\_back(headNeck);

segmentsCogVertices.push\_back(trunk);

segmentsCogVertices.push\_back(leftUpperArm);

segmentsCogVertices.push\_back(rightUpperArm);

segmentsCogVertices.push\_back(leftForeArm);

segmentsCogVertices.push\_back(rightForeArm);

segmentsCogVertices.push\_back(leftHand);

segmentsCogVertices.push\_back(rightHand);

segmentsCogVertices.push\_back(leftThigh);

segmentsCogVertices.push\_back(rightThigh);

segmentsCogVertices.push\_back(leftShank);

segmentsCogVertices.push\_back(rightShank);

segmentsCogVertices.push\_back(leftFoot);

segmentsCogVertices.push\_back(rightFoot);

glBindVertexArray(segmentsCogVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, segmentsCogVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(segmentsCogVertices[0]) \* segmentsCogVertices.size(), &segmentsCogVertices[0], GL\_DYNAMIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, segmentsCogVBO);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

comVertices.push\_back(bodyCOM);

glBindVertexArray(comVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, comVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(comVertices[0]) \* comVertices.size(), &comVertices[0], GL\_DYNAMIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, comVBO);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

}

void updateBvh()

{

if (frameChange)

{

bvhFrame++;

if (loop)

{

bvhFrame = bvhFrame % bvh->getNumFrames();

}

else if (!loop && (unsigned int)bvhFrame < bvh->getNumFrames())

{

}

else

{

frameChange = false;

bvhFrame = 0;

}

}

//std::cout << "move to " << frameto << std::endl;

bvh->moveTo(bvhFrame);

bvhVertices.clear();

bvhIndices.clear();

processBvh((Joint\*)bvh->getRootJoint(), bvhVertices, bvhIndices);

glBindBuffer(GL\_ARRAY\_BUFFER, bvhVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(bvhVertices[0]) \* bvhVertices.size(), &bvhVertices[0], GL\_DYNAMIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

}

/\*################################################################################################################################################\*/

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

};

unsigned int floorVBO, floorEBO, floorVAO;

glGenVertexArrays(1, &floorVAO);

glGenBuffers(1, &floorVBO);

glGenBuffers(1, &floorEBO);

glBindVertexArray(floorVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, floorVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(floorVertices), floorVertices, GL\_STATIC\_DRAW);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, floorEBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(floorIndices), floorIndices, GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 3 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(0);

Shader floorShader("floor.vs", "floor.fs");

// bvh

bvh = new Bvh2;

//bvh->load("data/example2.bvh");

const char\* filename = argv[1];

bvh->load(filename);

bvh->moveTo(bvhFrame);

bvhVertices.clear();

bvhIndices.clear();

processBvh((Joint\*)bvh->getRootJoint(), bvhVertices, bvhIndices);

bvhElements = (short)bvhIndices.size();

glGenVertexArrays(1, &segmentsCogVAO);

glGenBuffers(1, &segmentsCogVBO);

glGenVertexArrays(1, &comVAO);

glGenBuffers(1, &comVBO);

glGenVertexArrays(1, &bvhVAO);

glGenBuffers(1, &bvhVBO);

glGenBuffers(1, &bvhEBO);

glBindVertexArray(bvhVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, bvhVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(bvhVertices[0]) \* bvhVertices.size(), &bvhVertices[0], GL\_DYNAMIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, bvhVBO);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, bvhEBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(bvhIndices[0]) \* bvhIndices.size(), &bvhIndices[0], GL\_DYNAMIC\_DRAW);

Shader bvhShader("shader.vs", "shader.fs");

// ImGui Context

ImGui::CreateContext();

ImGuiIO& io = ImGui::GetIO();

(void)io;

ImGui::StyleColorsDark();

ImGui\_ImplGlfw\_InitForOpenGL(window, true);

ImGui\_ImplOpenGL3\_Init("#version 330");

bool showDemoWindow = true;

// mvp

glm::mat4 model = glm::mat4(1.0f);

glm::mat4 view = glm::mat4(1.0f);

glm::mat4 projection = glm::mat4(1.0f);

glm::mat4 mvp = glm::mat4(1.0f);

// scale the model if it's too big

//model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));

int graphFrames = bvh->getNumFrames() + 1;

// body com graph

std::vector<std::vector<float>> comGraph;

comGraph.resize(3);

comGraph[0].resize(graphFrames);

comGraph[1].resize(graphFrames);

comGraph[2].resize(graphFrames);

// head neck com graph

std::vector<std::vector<float>> headNeckGraph;

headNeckGraph.resize(3);

headNeckGraph[0].resize(graphFrames);

headNeckGraph[1].resize(graphFrames);

headNeckGraph[2].resize(graphFrames);

// trunk com graph

std::vector<std::vector<float>> trunkGraph;

trunkGraph.resize(3);

trunkGraph[0].resize(graphFrames);

trunkGraph[1].resize(graphFrames);

trunkGraph[2].resize(graphFrames);

// left upper arm com graph

std::vector<std::vector<float>> leftUpperArmGraph;

leftUpperArmGraph.resize(3);

leftUpperArmGraph[0].resize(graphFrames);

leftUpperArmGraph[1].resize(graphFrames);

leftUpperArmGraph[2].resize(graphFrames);

// right upper arm com graph

std::vector<std::vector<float>> rightUpperArmGraph;

rightUpperArmGraph.resize(3);

rightUpperArmGraph[0].resize(graphFrames);

rightUpperArmGraph[1].resize(graphFrames);

rightUpperArmGraph[2].resize(graphFrames);

// left fore arm com graph

std::vector<std::vector<float>> leftForeArmGraph;

leftForeArmGraph.resize(3);

leftForeArmGraph[0].resize(graphFrames);

leftForeArmGraph[1].resize(graphFrames);

leftForeArmGraph[2].resize(graphFrames);

// right fore arm com graph

std::vector<std::vector<float>> rightForeArmGraph;

rightForeArmGraph.resize(3);

rightForeArmGraph[0].resize(graphFrames);

rightForeArmGraph[1].resize(graphFrames);

rightForeArmGraph[2].resize(graphFrames);

// left hand com graph

std::vector<std::vector<float>> leftHandGraph;

leftHandGraph.resize(3);

leftHandGraph[0].resize(graphFrames);

leftHandGraph[1].resize(graphFrames);

leftHandGraph[2].resize(graphFrames);

// right hand com graph

std::vector<std::vector<float>> rightHandGraph;

rightHandGraph.resize(3);

rightHandGraph[0].resize(graphFrames);

rightHandGraph[1].resize(graphFrames);

rightHandGraph[2].resize(graphFrames);

// left thigh com graph

std::vector<std::vector<float>> leftThighGraph;

leftThighGraph.resize(3);

leftThighGraph[0].resize(graphFrames);

leftThighGraph[1].resize(graphFrames);

leftThighGraph[2].resize(graphFrames);

// right thigh com graph

std::vector<std::vector<float>> rightThighGraph;

rightThighGraph.resize(3);

rightThighGraph[0].resize(graphFrames);

rightThighGraph[1].resize(graphFrames);

rightThighGraph[2].resize(graphFrames);

// left shank com graph

std::vector<std::vector<float>> leftShankGraph;

leftShankGraph.resize(3);

leftShankGraph[0].resize(graphFrames);

leftShankGraph[1].resize(graphFrames);

leftShankGraph[2].resize(graphFrames);

// right shank com graph

std::vector<std::vector<float>> rightShankGraph;

rightShankGraph.resize(3);

rightShankGraph[0].resize(graphFrames);

rightShankGraph[1].resize(graphFrames);

rightShankGraph[2].resize(graphFrames);

// left foot com graph

std::vector<std::vector<float>> leftFootGraph;

leftFootGraph.resize(3);

leftFootGraph[0].resize(graphFrames);

leftFootGraph[1].resize(graphFrames);

leftFootGraph[2].resize(graphFrames);

// right foot com graph

std::vector<std::vector<float>> rightFootGraph;

rightFootGraph.resize(3);

rightFootGraph[0].resize(graphFrames);

rightFootGraph[1].resize(graphFrames);

rightFootGraph[2].resize(graphFrames);

double lastTimeFrame = glfwGetTime();

while (!glfwWindowShouldClose(window))

{

glfwPollEvents();

ImGui\_ImplOpenGL3\_NewFrame();

ImGui\_ImplGlfw\_NewFrame();

ImGui::NewFrame();

processInput(window);

glClearColor(backgroundColor[0], backgroundColor[1], backgroundColor[2], 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

float currentTime = (float)glfwGetTime();

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();

if (ImGui::Button("Reset"))

{

backgroundColor[0] = 0.2f;

backgroundColor[1] = 0.2f;

backgroundColor[2] = 0.2f;

floorColor[0] = 0.05f;

floorColor[1] = 0.05f;

floorColor[2] = 0.05f;

boneColor[0] = 0.5f;

boneColor[1] = 0.5f;

boneColor[2] = 0.5f;

jointColor[0] = 0.5f;

jointColor[1] = 1.0f;

jointColor[2] = 1.0f;

segmentComColor[0] = 1.0f;

segmentComColor[1] = 1.0f;

segmentComColor[2] = 0.0f;

comColor[0] = 1.0f;

comColor[1] = 0.0f;

comColor[2] = 0.0f;

boneWidth = 1.0;

jointPointSize = 2.0;

}

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);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Foot Mass Male", &footMassPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Foot Mass Female", &footMassPercent[1]);

ImGui::Columns(1);

ImGui::Separator();

ImGui::Text(" ");

ImGui::Columns(1);

ImGui::Separator();

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

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Head & Neck Length Male", &headNeckLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Head & Neck Length Female", &headNeckLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Trunk Length Male", &trunkLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Trunk Length Female", &trunkLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Upper Arm Length Male", &upperArmLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Upper Arm Length Female", &upperArmLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Fore Arm Length Male", &foreArmLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Fore Arm Length Female", &foreArmLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Hand Length Male", &handLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Hand Length Female", &handLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Thigh Length Male", &thighLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Thigh Length Female", &thighLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Shank Length Male", &shankLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Shank Length Female", &shankLengthPercent[1]);

ImGui::Columns(1);

ImGui::Columns(2);

ImGui::Separator();

ImGui::InputFloat("Foot Length Male", &footLengthPercent[0]);

ImGui::NextColumn();

ImGui::InputFloat("Foot Length Female", &footLengthPercent[1]);

ImGui::Columns(1);

ImGui::Separator();

ImGui::Text(" ");

}

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

{

comGraph[0][bvhFrame] = comVertices[0].x;

comGraph[1][bvhFrame] = comVertices[0].y;

comGraph[2][bvhFrame] = comVertices[0].z;

static float comGraphXHeight = 150.0f;

static float comGraphYHeight = 150.0f;

static float comGraphZHeight = 150.0f;

ImGui::PlotHistogram("Body COM X", &comGraph[0][0], graphFrames, 0, "", -comGraphXHeight, comGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Body COM X Height", &comGraphXHeight, 1, 200);

ImGui::PlotHistogram("Body COM Y", &comGraph[1][0], graphFrames, 0, "", -comGraphYHeight, comGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Body COM Y Height", &comGraphYHeight, 1, 200);

ImGui::PlotHistogram("Body COM Z", &comGraph[2][0], graphFrames, 0, "", -comGraphZHeight, comGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Body COM Z Height", &comGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Head & Neck COM"))

{

headNeckGraph[0][bvhFrame] = segmentsCogVertices[0].x;

headNeckGraph[1][bvhFrame] = segmentsCogVertices[0].y;

headNeckGraph[2][bvhFrame] = segmentsCogVertices[0].z;

static float headNeckGraphXHeight = 150.0f;

static float headNeckGraphYHeight = 150.0f;

static float headNeckGraphZHeight = 150.0f;

ImGui::PlotHistogram("Head Neck COM X", &headNeckGraph[0][0], graphFrames, 0, "", -headNeckGraphXHeight, headNeckGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Head Neck COM X Height", &headNeckGraphXHeight, 1, 200);

ImGui::PlotHistogram("Head Neck COM Y", &headNeckGraph[1][0], graphFrames, 0, "", -headNeckGraphYHeight, headNeckGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Head Neck COM Y Height", &headNeckGraphYHeight, 1, 200);

ImGui::PlotHistogram("Head Neck COM Z", &headNeckGraph[2][0], graphFrames, 0, "", -headNeckGraphZHeight, headNeckGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Head Neck COM Z Height", &headNeckGraphZHeight, 1, 200);

}

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

{

trunkGraph[0][bvhFrame] = segmentsCogVertices[1].x;

trunkGraph[1][bvhFrame] = segmentsCogVertices[1].y;

trunkGraph[2][bvhFrame] = segmentsCogVertices[1].z;

static float trunkGraphXHeight = 150.0f;

static float trunkGraphYHeight = 150.0f;

static float trunkGraphZHeight = 150.0f;

ImGui::PlotHistogram("Trunk COM X", &trunkGraph[0][0], graphFrames, 0, "", -trunkGraphXHeight, trunkGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Trunk COM X Height", &trunkGraphXHeight, 1, 200);

ImGui::PlotHistogram("Trunk COM Y", &trunkGraph[1][0], graphFrames, 0, "", -trunkGraphYHeight, trunkGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Trunk COM Y Height", &trunkGraphYHeight, 1, 200);

ImGui::PlotHistogram("Trunk COM Z", &trunkGraph[2][0], graphFrames, 0, "", -trunkGraphZHeight, trunkGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Trunk COM Z Height", &trunkGraphZHeight, 1, 200);

}

if (ImGui::CollapsingHeader("Left Upper Arm COM"))

{

leftUpperArmGraph[0][bvhFrame] = segmentsCogVertices[2].x;

leftUpperArmGraph[1][bvhFrame] = segmentsCogVertices[2].y;

leftUpperArmGraph[2][bvhFrame] = segmentsCogVertices[2].z;

static float leftUpperArmGraphXHeight = 150.0f;

static float leftUpperArmGraphYHeight = 150.0f;

static float leftUpperArmGraphZHeight = 150.0f;

ImGui::PlotHistogram("Left Upper Arm COM X", &leftUpperArmGraph[0][0], graphFrames, 0, "", -leftUpperArmGraphXHeight, leftUpperArmGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Upper Arm COM X Height", &leftUpperArmGraphXHeight, 1, 200);

ImGui::PlotHistogram("Left Upper Arm COM Y", &leftUpperArmGraph[1][0], graphFrames, 0, "", -leftUpperArmGraphYHeight, leftUpperArmGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Left Upper Arm COM Y Height", &leftUpperArmGraphYHeight, 1, 200);

ImGui::PlotHistogram("Left Upper Arm COM Z", &leftUpperArmGraph[2][0], graphFrames, 0, "", -leftUpperArmGraphZHeight, leftUpperArmGraphZHeight, ImVec2(0, 100), 4);

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

}

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

{

rightUpperArmGraph[0][bvhFrame] = segmentsCogVertices[3].x;

rightUpperArmGraph[1][bvhFrame] = segmentsCogVertices[3].y;

rightUpperArmGraph[2][bvhFrame] = segmentsCogVertices[3].z;

static float rightUpperArmGraphXHeight = 150.0f;

static float rightUpperArmGraphYHeight = 150.0f;

static float rightUpperArmGraphZHeight = 150.0f;

ImGui::PlotHistogram("Right Upper Arm COM X", &rightUpperArmGraph[0][0], graphFrames, 0, "", -rightUpperArmGraphXHeight, rightUpperArmGraphXHeight, ImVec2(0, 100), 4);

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

ImGui::PlotHistogram("Right Upper Arm COM Y", &rightUpperArmGraph[1][0], graphFrames, 0, "", -rightUpperArmGraphYHeight, rightUpperArmGraphYHeight, ImVec2(0, 100), 4);

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

ImGui::PlotHistogram("Right Upper Arm COM Z", &rightUpperArmGraph[2][0], graphFrames, 0, "", -rightUpperArmGraphZHeight, rightUpperArmGraphZHeight, ImVec2(0, 100), 4);

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

}

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

{

leftForeArmGraph[0][bvhFrame] = segmentsCogVertices[4].x;

leftForeArmGraph[1][bvhFrame] = segmentsCogVertices[4].y;

leftForeArmGraph[2][bvhFrame] = segmentsCogVertices[4].z;

static float leftForeArmGraphXHeight = 150.0f;

static float leftForeArmGraphYHeight = 150.0f;

static float leftForeArmGraphZHeight = 150.0f;

ImGui::PlotHistogram("Left Fore Arm COM X", &leftForeArmGraph[0][0], graphFrames, 0, "", -leftForeArmGraphXHeight, leftForeArmGraphXHeight, ImVec2(0, 100), 4);

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

ImGui::PlotHistogram("Left Fore Arm COM Y", &leftForeArmGraph[1][0], graphFrames, 0, "", -leftForeArmGraphYHeight, leftForeArmGraphYHeight, ImVec2(0, 100), 4);

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

ImGui::PlotHistogram("Left Fore Arm COM Z", &leftForeArmGraph[2][0], 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;

static float rightShankGraphXHeight = 150.0f;

static float rightShankGraphYHeight = 150.0f;

static float rightShankGraphZHeight = 150.0f;

ImGui::PlotHistogram("Right Shank COM X", &rightShankGraph[0][0], graphFrames, 0, "", -rightShankGraphXHeight, rightShankGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Shank COM X Height", &rightShankGraphXHeight, 1, 200);

ImGui::PlotHistogram("Right Shank COM Y", &rightShankGraph[1][0], graphFrames, 0, "", -rightShankGraphYHeight, rightShankGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Shank COM Y Height", &rightShankGraphYHeight, 1, 200);

ImGui::PlotHistogram("Right Shank COM Z", &rightShankGraph[2][0], graphFrames, 0, "", -rightShankGraphZHeight, rightShankGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Shank COM Z Height", &rightShankGraphZHeight, 1, 200);

}

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

{

leftFootGraph[0][bvhFrame] = segmentsCogVertices[12].x;

leftFootGraph[1][bvhFrame] = segmentsCogVertices[12].y;

leftFootGraph[2][bvhFrame] = segmentsCogVertices[12].z;

static float leftFootGraphXHeight = 150.0f;

static float leftFootGraphYHeight = 150.0f;

static float leftFootGraphZHeight = 150.0f;

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

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);

}

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

{

rightFootGraph[0][bvhFrame] = segmentsCogVertices[12].x;

rightFootGraph[1][bvhFrame] = segmentsCogVertices[12].y;

rightFootGraph[2][bvhFrame] = segmentsCogVertices[12].z;

static float rightFootGraphXHeight = 150.0f;

static float rightFootGraphYHeight = 150.0f;

static float rightFootGraphZHeight = 150.0f;

ImGui::PlotHistogram("Right Foot COM X", &rightFootGraph[0][0], graphFrames, 0, "", -rightFootGraphXHeight, rightFootGraphXHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Foot COM X Height", &rightFootGraphXHeight, 1, 200);

ImGui::PlotHistogram("Right Foot COM Y", &rightFootGraph[1][0], graphFrames, 0, "", -rightFootGraphYHeight, rightFootGraphYHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Foot COM Y Height", &rightFootGraphYHeight, 1, 200);

ImGui::PlotHistogram("Right Foot COM Z", &rightFootGraph[2][0], graphFrames, 0, "", -rightFootGraphZHeight, rightFootGraphZHeight, ImVec2(0, 100), 4);

ImGui::SliderFloat("Right Foot COM Z Height", &rightFootGraphZHeight, 1, 200);

}

ImGui::End();

}

if (showDemoWindow)

ImGui::ShowDemoWindow(&showDemoWindow);

ImGui::Render();

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

glfwSwapBuffers(window);

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

{

Sleep(10);

}

lastTimeFrame += 1.0 / FPS;

}

glfwTerminate();

return 0;

}

/\*################################################################################################################################################\*/

// GLFW callbacks definitions

void frameBufferSizeCallback(GLFWwindow\* window, int width, int height)

{

glViewport(0, 0, width, height);

}

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

if (glfwGetKey(window, GLFW\_KEY\_LEFT\_SHIFT) == GLFW\_PRESS)

cameraSpeed = 300.0f;

else

cameraSpeed = 150.0f;

float appliedSpeed = cameraSpeed \* deltaTime;

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

cameraPos += appliedSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

cameraPos -= appliedSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)

cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* appliedSpeed;

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)

cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* appliedSpeed;

if (glfwGetKey(window, GLFW\_KEY\_E) == GLFW\_PRESS)

cameraPos += appliedSpeed \* cameraUp;

if (glfwGetKey(window, GLFW\_KEY\_Q) == GLFW\_PRESS)

cameraPos -= appliedSpeed \* cameraUp;

if (glfwGetKey(window, GLFW\_KEY\_O) == GLFW\_PRESS)

FPS -= 1;

if (glfwGetKey(window, GLFW\_KEY\_P) == GLFW\_PRESS)

FPS += 1;

}

void mouseCallback(GLFWwindow\* window, double xpos, double ypos)

{

float xposf = (float)xpos;

float yposf = (float)ypos;

int state = glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_RIGHT);

if (state == GLFW\_PRESS)

{

if (firstMouse)

{

lastX = xposf;

lastY = yposf;

firstMouse = false;

}

float xoffset = xposf - lastX;

float yoffset = lastY - yposf;

lastX = xposf;

lastY = yposf;

float sensitivity = 0.1f;

xoffset \*= sensitivity;

yoffset \*= sensitivity;

yaw += xoffset;

pitch += yoffset;

if (pitch > 89.0f)

pitch = 89.0f;

if (pitch < -89.0f)

pitch = -89.0f;

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

cameraFront = glm::normalize(front);

}

}

**Lampiran 6**

// shader.vs

#version 330 core

layout (location = 0) in vec4 aPos;

uniform mat4 mvp;

void main()

{

gl\_Position = mvp \* aPos;

};

**Lampiran 7**

// shader.fs

#version 330 core

out vec4 FragColor;

uniform vec3 ourColor;

void main()

{

FragColor = vec4(ourColor, 1.0);

};

**Lampiran 8**

// floor.vs

#version 330 core

layout (location = 0) in vec3 aPos;

uniform mat4 mvp;

void main()

{

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

}

**Lampiran 9**

// floor.fs

#version 330 core

out vec4 FragColor;

uniform vec3 ourColor;

void main()

{

FragColor = vec4(ourColor, 1.0);

};