// ... includes, function prototypes, etc.

// Really just a local coordinate system defined by a translation, rotation, and

// scale relative to the parent joint's coordinate system.

struct JointPose

{

JointPose\* parent;

color4 color;

vec2 translation;

float rotation;

float scale;

};

// In this case, there are 7 joints in our skeleton

struct Pose

{

JointPose joints[7];

};

// Here is the vertex shader. It should be loaded into a string but shown here as a comment for clarity.

const std::string vertex\_shader\_source = ... /\*

#version 330

uniform mat4 bind\_pose\_inv[7];

uniform mat4 current\_pose[7];

uniform vec4 current\_pose\_colors[7];

layout(location = 0) in vec2 position;

layout(location = 1) in uint joint\_0\_index;

layout(location = 2) in uint joint\_1\_index;

layout(location = 3) in uint joint\_2\_index;

layout(location = 4) in float joint\_0\_weight;

layout(location = 5) in float joint\_1\_weight;

layout(location = 6) in float joint\_2\_weight;

out vec4 color;

void main()

{

vec4 vertex\_coords = vec4(position, 0, 1);

gl\_Position = vec4(0,0,0,0);

color = vec4(0,0,0,0);

// First joint affecting vertex

color += joint\_0\_weight \* current\_pose\_colors[joint\_0\_index];

gl\_Position += joint\_0\_weight \* (current\_pose[joint\_0\_index] \*

bind\_pose\_inv[joint\_0\_index] \*

vertex\_coords);

// ... Repeat for other joints affecting vertex.

}

\*/

// Here is the fragment shader. It should be loaded into a string but shown here as a comment.

const std::string fragment\_shader\_source = ... /\*

#version 330

in vec4 color;

layout(location = 0) out vec4 out\_fragcolor;

void main()

{

out\_fragcolor = color;

}

\*/

struct Vertex

{

vec2 position; ///< The vertex's 2D position in bind-pose model space.

GLuint joint\_indices[3]; ///< The indices of 3 joints which affect the vertex.

GLfloat joint\_weights[3]; ///< The amount that the joints identified above affect the vertex.

};

// A skeletal mesh object is a Vertex Array Object (VAO) that has an Index Buffer Object (IBO) and a Vertex Buffer

// Object (VBO) which is suitable for use with a skinning vertex shader.

struct SkeletalMesh

{

SkeletalMesh()

{

glGenVertexArrays(1, &vao\_id\_); // Create VAO

glGenBuffers(1, &vbo\_id\_); // Create VBO

glGenBuffers(1, &ibo\_id\_); // Create IBO

}

~SkeletalMesh() { /\* ... \*/ }

void uploadMesh() const

{

glBindVertexArray(vao\_id); // bind VAO

glBindBuffer(GL\_ARRAY\_BUFFER, vbo\_id); // bind VBO

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, ibo\_id); // bind IBO

glBufferData(GL\_ARRAY\_BUFFER, vertices.size() \* sizeof(Vertex), vertices.data(), GL\_STATIC\_DRAW);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, indices.size() \* sizeof(GLushort), indices.data(), GL\_STATIC\_DRAW);

void\* attr\_ptr\_position = nullptr;

void\* attr\_ptr\_bone\_index\_0 = (void\*)(sizeof(vec2));

void\* attr\_ptr\_bone\_index\_1 = (void\*)(sizeof(vec2) + sizeof(GLuint));

void\* attr\_ptr\_bone\_index\_2 = (void\*)(sizeof(vec2) + sizeof(GLuint) \* 2);

void\* attr\_ptr\_bone\_weight\_0 = (void\*)(sizeof(vec2) + sizeof(GLuint) \* 3);

void\* attr\_ptr\_bone\_weight\_1 = (void\*)(sizeof(vec2) + sizeof(GLuint) \* 3 + sizeof(GLfloat));

void\* attr\_ptr\_bone\_weight\_2 = (void\*)(sizeof(vec2) + sizeof(GLuint) \* 3 + sizeof(GLfloat) \* 2);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), attr\_ptr\_position);

glVertexAttribIPointer(1, 1, GL\_UNSIGNED\_INT, sizeof(Vertex), attr\_ptr\_bone\_index\_0);

glVertexAttribIPointer(2, 1, GL\_UNSIGNED\_INT, sizeof(Vertex), attr\_ptr\_bone\_index\_1);

glVertexAttribIPointer(3, 1, GL\_UNSIGNED\_INT, sizeof(Vertex), attr\_ptr\_bone\_index\_2);

glVertexAttribPointer(4, 1, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), attr\_ptr\_bone\_weight\_0);

glVertexAttribPointer(5, 1, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), attr\_ptr\_bone\_weight\_1);

glVertexAttribPointer(6, 1, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), attr\_ptr\_bone\_weight\_2);

glEnableVertexAttribArray(0);

// ...

glEnableVertexAttribArray(6);

// GL\_ARRAY\_BUFFER is not part of the VAO state, so we need to make sure we unbind the VBO.

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0); // un-bind VAO

}

std::vector<Vertex> vertices;

std::vector<GLushort> indices;

GLuint vao\_id\_;

GLuint vbo\_id\_;

GLuint ibo\_id\_;

};

SkeletalMesh\* mesh;

const size\_t N\_POSES = 3; ///< The number of different skeleton poses we have available.

Pose poses[N\_POSES]; ///< An array of skeleton poses.

Pose current\_pose;

// Generates and returns a matrix which transforms coordinates from a joint's local coordinate space to the joint's

// parent's coordinate space.

mat4 getJointLocalTransform(const JointPose& joint\_pose)

{

mat4 transform;

transform = glm::translate(transform, vec3(joint\_pose.translation, 0));

if (joint\_pose.rotation != 0.0f)

transform = glm::rotate(transform, joint\_pose.rotation, vec3(0, 0, 1));

if (joint\_pose.scale != 1.0f)

transform = glm::scale(transform, vec3(joint\_pose.scale,

joint\_pose.scale,

joint\_pose.scale));

return transform;

}

// Generates and returns a matrix which transforms coordinates from a joint's local coordinate space to model space.

mat4 getJointTransform(const JointPose& joint\_pose)

{

// if this joint has a parent, then lets start in it's

// coordinate space, otherwise start with an identity matrix.

if (joint\_pose.parent != nullptr)

return getJointTransform(\*joint\_pose.parent) \* getJointLocalTransform(joint\_pose);

return getJointLocalTransform(joint\_pose);

}

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

{

// freeglut initialization

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(800, 800);

glutCreateWindow("Skeletal Mesh Skinning Demo");

// GLEW initialization

GLenum err = glewInit();

if (err != GLEW\_OK)

{

std::cerr << "Error intitializing GLEW!" << std::endl;

return 0;

}

initPoses();

initGL();

glutReshapeFunc(reshape);

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutPassiveMotionFunc(mouseMove);

glutMotionFunc(mouseMove);

glutMainLoop();

cleanup();

return 0;

}

void initGL() { /\* ... \*/ }

void initShaderProgram() { /\* ... \*/ }

void initMeshes() { /\* ... \*/ }

void initPoses() { /\* ... \*/ }

void cleanup() { /\* ... \*/ }

void reshape(int width, int height) { /\* ... \*/ }

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(shader\_program\_id);

glBindVertexArray(mesh->vao\_id);

mat4 current\_pose\_data[7];

vec4 current\_pose\_color\_data[7];

for (size\_t joint = 0; joint < 7; ++joint)

{

JointPose& jp = current\_pose.joints[joint];

current\_pose\_data[joint] = getJointTransform(jp);

current\_pose\_color\_data[joint] = jp.color;

}

glUniformMatrix4fv(current\_pose\_uniform\_location, 7, GL\_FALSE, &current\_pose\_data[0][0][0]);

glUniform4fv(current\_pose\_colors\_uniform\_location, 7, &current\_pose\_color\_data[0][0]);

glDrawElements(GL\_TRIANGLES, mesh->indices.size(), GL\_UNSIGNED\_SHORT, 0);

glBindVertexArray(0);

glUseProgram(0);

glutSwapBuffers();

}

void keyboard(unsigned char key, int x, int y) { /\* ... \*/ }

void mouseMove(int x, int y)

{

float f = float(x) / viewport.x;

float g = 1 - f;

current\_pose = poses[left\_pose];

for (size\_t joint = 0; joint < 7; ++joint)

{

JointPose& jp = current\_pose.joints[joint];

if (jp.parent)

{

ptrdiff\_t offset = reinterpret\_cast<char\*>(jp.parent) - reinterpret\_cast<char\*>(&poses[left\_pose]);

jp.parent = reinterpret\_cast<JointPose\*>(reinterpret\_cast<char\*>(&current\_pose) + offset);

}

current\_pose.joints[joint].color = poses[left\_pose].joints[joint].color \* g +

poses[right\_pose].joints[joint].color \* f;

current\_pose.joints[joint].rotation = poses[left\_pose].joints[joint].rotation \* g +

poses[right\_pose].joints[joint].rotation \* f;

current\_pose.joints[joint].scale = poses[left\_pose].joints[joint].scale \* g +

poses[right\_pose].joints[joint].scale \* f;

current\_pose.joints[joint].translation = poses[left\_pose].joints[joint].translation \* g +

poses[right\_pose].joints[joint].translation \* f;

}

glutPostRedisplay();

}