EXTENDING SAGE: A SIMPLE ADAPTABLE GAME ENGINE

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Morgan Darke

SPRING
2013
EXTENDING SAGE: A SIMPLE ADAPTABLE GAME ENGINE

A Project

by

Morgan Darke

Approved by:

__________________________________, Committee Chair
John Clevenger, Ph.D.

__________________________________, Second Reader
V. Scott Gordon, Ph.D.

__________________________________
Date

iii
Student: Morgan Darke

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator
Behnam Arad, Ph.D.

______________________________
Date

Department of Computer Science
Abstract

of

EXTENDING SAGE: A SIMPLE ADAPTABLE GAME ENGINE

by

Morgan Darke

The computer science department at CSUS offers a class on computer game architecture as an elective. In this course, students are provided with a simple game engine in which to extend or modify. This game engine is known as SAGE: a simple adaptable game engine. SAGE provides some basic functionalities of a game engine, but it lacks support for many features found in modern computer games.

The purpose of this project is to enhance the capabilities of SAGE by implementing three key features that would allow students to develop games that are more advanced. These features include support for loading models in OgreXML format, support for skeletal animation, and support for rendering models using GLSL shader programs.

_______________________, Committee Chair
John Clevenger, Ph.D.

_______________________
Date

v
ACKNOWLEDGEMENTS

I would like to thank my parents who have always supported me throughout my academic career, regardless of how long it took.

I would like to thank my fiancée, Mimi Xue, for always being there to motivate me and not letting me succumb to procrastination.

Finally, I would like to thank my advisor, Dr. John Clevenger, whose classes were responsible for introducing me to computer graphics.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Acknowledgements</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>x</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. RELEVANT TOPICS AND TECHNOLOGIES</td>
<td>3</td>
</tr>
<tr>
<td>2.1 SAGE</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Model Loading</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Skeletal Animation</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Shader Programs</td>
<td>5</td>
</tr>
<tr>
<td>3. IMPLEMENTATION</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Model Loading</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Skeletal Animation</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Shader Programs</td>
<td>10</td>
</tr>
<tr>
<td>4. USAGE</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Model Loading</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Skeletal Animation</td>
<td>14</td>
</tr>
<tr>
<td>4.3 Shader Programs</td>
<td>16</td>
</tr>
<tr>
<td>5. FUTURE WORK</td>
<td>18</td>
</tr>
<tr>
<td>5.1 Model Loading</td>
<td>18</td>
</tr>
</tbody>
</table>

vii
5.2 Key-frame Interpolation ................................................................. 18
5.3 GPU Skinning .................................................................................. 19
Appendix A. Source Code in Java ................................................................. 20
Appendix B. OgreXML Document Type Definitions ...................................... 89
References ................................................................................................ 94
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1  An example model loaded from WoWModelViewer</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2.2  The 3D Graphics Pipeline</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3.1  Transformation for animating a vertex</td>
<td>10</td>
</tr>
<tr>
<td>Figure 4.1  Example of loading an animated OgreXML model</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4.2  Example of loading an OgreXML model without animation data</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4.3  Example of starting an animation sequence</td>
<td>15</td>
</tr>
<tr>
<td>Figure 4.4  Example of updating a model’s current animation</td>
<td>15</td>
</tr>
<tr>
<td>Figure 4.5  Example of creating a GLSL shader program</td>
<td>16</td>
</tr>
<tr>
<td>Figure 4.6  Example of attaching a shader program to a hierarchical model</td>
<td>16</td>
</tr>
<tr>
<td>Figure 4.7  Example of creating uniform variables</td>
<td>17</td>
</tr>
</tbody>
</table>
LIST OF ABBREVIATIONS

1. SAGE: Simple Adaptable Game Engine
2. OpenGL: Open Graphics Language
3. XML: Extensible Markup Language
4. GLSL: OpenGL Shading Language
5. jME: jMonkeyEngine
Chapter 1

INTRODUCTION

Students enrolled in CSC 165, Computer Game Architecture and Implementation, are assigned with creating a playable computer game while simultaneously developing a basic game engine. A simple, modular game engine known as SAGE[1] is provided as a foundation to start developing their own game engine. While SAGE is sufficient for very simple games, it is lacking support for many features commonly found in contemporary video games. Furthermore, some of the advanced game design concepts taught, such as animation, are not presented to the students until the class is almost over. Due to the time constraints that a one semester long class presents, students are unable to implement more advanced features themselves. This project identifies three essential features that SAGE is lacking and integrates them into the game engine.

A small subset of features found in all modern computer games are the ability to load and render models, support animation, and replace the fixed function render pipeline with programmable shaders. Artists create three dimensional models in modeling programs such as Blender[2] or Maya[3] which can then be imported into games in a variety of file formats. In this project, I decided on focusing on models exported in XML[4] because it is a portable format which contains human-readable data.

Many models also include animation information. Animation sequences are often exported as key-frames, in which the position of a model is defined for specific points in time. The model's position in between frames is determined by an interpolation of the two nearest key-frames. Shaders are computer programs specially designed to run on a graphics
processor[5]. This project adds support for shaders to replace the portion of the fixed function pipeline that handles vertex transformations and per-fragment operations.

The final deliverable for this project is a software application that renders a three-dimensional scene using the SAGE game engine. The scene displayed contains multiple animated models loaded from an external 3D modeling program and is rendered with customized shader programs.
Chapter 2

RELEVANT TOPICS AND TECHNOLOGIES

2.1 SAGE

SAGE is a simple game engine based on the open source jMonkeyEngine[6], a full featured game engine written entirely in Java[7]. Unfortunately, due to the enormity of the jME code base, it is infeasible for students to modify the inner-workings of the engine without a lot of in-depth knowledge. SAGE was developed with a modular design where it would be easy to extend or override the game engine’s functionalities.

In the beginning of the semester, students do not yet have the knowledge needed to begin working on their own game engine; they have to rely completely on SAGE. As the semester progresses, students begin replacing portions of SAGE, such as the camera and rendering systems, with their own implementations. The problem is that the functionality offered by SAGE is extremely limiting. There is no support for animations, which are required for even the most basic games. SAGE only supports a single primitive file format to load external models, and does not support animation data or hierarchical meshes. SAGE relies on the fixed function pipeline for rendering, which has been replaced by programmable shaders.

2.2 Model Loading

Many three-dimensional modeling programs are available which allow users to create their own digital content. However, most applications have their own proprietary format of storing model data. An application-independent way of describing model data was needed, leading to several standardized formats. XML is widely used as a platform-independent method of describing or transferring data including three-dimensional models. This project focuses on loading models...
from OgreXML[8], but other XML formats, such as COLLADA[9], are also used in the graphics industry.

Graphics processors cannot directly render geometry from model files. Instead, the information stored in the file needs to be loaded into a series of buffers beforehand. A model can be rendered with a minimum of a single buffer containing vertex locations; however, most models include data for texture coordinates, normals, and vertex indices as well. Model loading is the process of extracting data from the model file and populating the necessary buffers so that the geometry can be drawn by the graphics hardware.

Figure 2.1 An example model loaded from WoWModelViewer[10]
2.3 Skeletal Animation

Animating a character model described as a polygon mesh by moving each vertex in the mesh is impractical. It is more convenient to specify the motion of characters through the movement of an internal articulated skeleton from which the movement of the surrounding polygon may then be deduced[11]. The skeleton is comprised of a hierarchical structure of bones, or joints, in which each joint’s initial position and rotation is described relative to its parent joint. At the top of the hierarchy lies the root joint, also known as the spine, which is described relative to the origin of the character’s coordinate system.

The digital artist defines key-frames which represent the orientation of each of the joints at specific points in time. The game engine is responsible for inferring the configuration for the model’s joints in between key-frames. The more key-frames a model defines for an animation sequence, the more natural it will appear, at the cost of having a larger memory footprint. Each vertex of the model is attached to one or more joints. Translating or rotating a joint will also move the vertices it influences. Animation is done by controlling the movement of the model’s joints, which in turn manipulate the model’s vertices.

2.4 Shader Programs

In modern graphics hardware, most of the fixed function graphics pipeline has been replaced by programmable shader stages. The programmable shader stages include the vertex shader, tessellation stages, geometry shader, and pixel shader, as shown in Figure 2.2. Additionally, compute shaders extend the 3D pipeline beyond graphics to support general purpose GPU computing[12]. A shader program is one or more shader stages linked together to form an
executable binary which can run on the graphics hardware. This project adds support for vertex and pixel shaders, leaving the other shader stages to be implemented as part of future work.

Figure 2.2. The 3D Graphics Pipeline[12]
Chapter 3

IMPLEMENTATION

3.1 Model Loading

SAGE already provides a model loader for external models saved in Wavefront OBJ file format[13]. While this format is sufficient for simple models, it lacks support for hierarchical meshes and animation data. Additionally, the OBJ model loader in SAGE does not implement all of the commands supported by the file format. OgreXML was chosen as a new format to be supported by SAGE because it overcomes the limitations of the OBJ file format.

Since OgreXML is an XML based file format, a parser needed to be written to extract the information in XML and convert it into an internal representation that SAGE could use. One problem encountered during the design of a parser for the OgreXML file format was that there is no official schema for the datatypes used. Not only would the parser need to be able to extract the data to create the model, but it would have to distinguish between valid and invalid files without Java’s automatic XML validation. However, Ogre[14] does publish a Document Type Definition for its XML formats, which defines the structure of each element and where they appear in the document[15].

The model loader is implemented as a stack-based recursive descent parser. The stack is used as a method to enforce well-formedness by ensuring each start element has a matching end element. When the parser encounters an XML StartElement event, the element is pushed on top on the stack. If the current element is a terminal then the primitive is constructed from the data and returned, otherwise the parser searches for a valid following element. When an XML EndElement
event is encountered, the stack is popped and the element popped is compared to the current
element. The parsing of the model is finished once the stack is empty.

OgreXML exports models into three separate files: one file contains the geometry data, another
file contains material information, and the last contains animation data. The file containing the
model’s geometry is known as the mesh, and contains vertex, normal, texture coordinate, and
index information for the model. Additionally, in the case of hierarchical models, the mesh also
details how to construct each of the sub-meshes. The mesh also contains links to the material and
skeleton files if the model uses any materials or has any animations.

3.2 Skeletal Animation

When an OgreXML model with animation data is loaded, the joint hierarchy is initialized to
calculate the transformations needed to transform a vertex from object space to joint space. This
is done by starting at the root of the hierarchy and for each child concatenating transformations of
the parent’s to-model-space transformation with the child’s from-parent-space transformation.
Joint transformations are stored as a four-by-four matrix containing values for translation,
rotation and scaling. Because transformations are stored as matrices, concatenation of
transformations is simply a multiplication of the two matrices.

The animation sequences that OgreXML models are exported with are given names, such as
“Attack” or “Fly”. Each model keeps a mapping of animation sequence names to a collection of
key-frame data, along with a current animation, and a current time inside the animating track.
When the update method of the game engine is called, models need their animation sequences
updated as well. The update method receives a time difference, usually measured in milliseconds, by which to increment the current animation time.

When it is time to animate the model, the game engine needs to calculate the set of key-frames, closest to the current animation time, that describe how each joint is oriented. Just as joints are described in relation to their parent joint, key-frames also represent joint orientation relative to the parent joint. Each time a new set of key-frames is to be constructed, the entire joint hierarchy needs to be traversed in order to compute the orientations relative to model space.

Once all the transformations have been computed, new buffers need to be created containing the animated vertex locations and animated normals. Animated vertex locations are calculated by left multiplying the original vertex by the inverse from-model-space-to-joint-space matrix and then left multiplied again by the animation-to-model-space matrix. This calculation is illustrated in Figure 3.1.

In order for lighting calculations to be correct during animation, the model’s normals must also be transformed. When a linear transformation $M$ is performed on a mesh, the normal must be transformed by the inverse transpose of $M[16]$. In models that are more complicated, each vertex can have multiple joints affecting its animation. In such cases, the resulting animated vertex or normal from each contributing joint is multiplied by a weight, and the final value is determined by using a weighted sum.
In a commercial game engine, animation time between key-frames is determined by interpolating between the closest two key-frames. SAGE’s implementation does not support interpolation, and instead finds the closest key-frame less than or equal to the current animation time. The reason for this is that linear interpolation of the rotation portion of the transformation matrix produces irregular angular velocities. The correct way to interpolate key-frames would be to represent the transformations as translation vectors and rotation quaternions. The resulting translation can be found by linearly interpolating the vectors while the rotation can be calculated by using spherical linear interpolation (SLERP) on the quaternions. Due to time constraints, I was not able to integrate key-frame interpolation into this project.

3.3 Shader Programs

SAGE relies on features long deprecated since the release of OpenGL 3.0 in 2008. Deprecated features include immediate-mode rendering and all fixed function vertex and fragment processing[18]. Modern applications using OpenGL are required to use shader programs for all of
their rendering. The decision to use such outdated implementations in SAGE comes from the fact that students do not need to have taken any OpenGL courses as prerequisites and using the fixed function pipeline for rendering is intuitive and easy to learn.

OpenGL supports several different shading languages including GLSL, NVIDIA’s Cg[19], and ARB assembly language[20]. This project adds support for only shaders written in GLSL, as it is the only language that is part of the official OpenGL specification as of version 2.0[21]. To create a shader program, source files must be compiled and linked by the graphics driver into an executable program capable of running on the graphics hardware. SAGE implements shader programs as GLSLShaderProgram objects that encapsulate the shader source and mechanisms for compiling, linking, and error handling.

Applications can pass data to shader programs through the use of uniform variables. Uniform variables are like constant global variables that retain the same value throughout a particular draw call. Each uniform is associated with a data type and a location within the shader program. Uniform types consist of floating point, integer, or Boolean data in scalar, vector, and matrix format. The uniform’s location is used to set the value of the variable through the OpenGL API calls of the form `glUniform*`.

Each GLSLShaderProgram object contains a mapping of names to GLSLUniform objects. Once the shader program has been created, the user needs to attach any GLSLUniform objects that are going to be used to the shader program and set their initial values. Failure to do so will result in the shader using default values, generally zero, for any uninitialized uniform variables. During the update loop in the game engine, it is possible for the application to change the values of uniform
variables in any shader program. These changes are not applied immediately because there is no guarantee that the modified shader program is the one currently active nor is there an active GL context to make the appropriate API calls. Instead, when the renderer is about to draw a model using a shader program, it first applies all of its uniform variables to ensure any changes that were made during the update loop take place.
4.1 Model Loading

The following code snippets demonstrate how to load models from OgreXML files into SAGE. An OgreXMLParser object needs to be constructed once, and is able to be reused if multiple models are to be loaded. The `loadModel` method takes three filenames as Strings as parameters. The first parameter specifies which file contains the geometry data, also known as the mesh, for the model. The second parameter specifies the material file, which contains filenames for any texture files the model uses. The final parameter specifies which file contains animation data for the model.

The `loadModel` method returns a Group SceneNode, which represents a hierarchy of Model3DTriMesh objects. Each submesh shares geometry with its sibling nodes, but can contain its own texture or shader program. Manipulating the root of the hierarchy via translation, rotation, or scaling will also perform the transformation on each of the sub-meshes. Loading models must be enclosed within a try-catch block in order to recover from any errors encountered during the parsing process.
4.2 Skeletal Animation

Each of the sub-meshes in the Group hierarchy is an instance of a Model3DTriMesh object. Model3DTriMeshes are specialized TriMesh objects that have methods to handle animations. These methods consist of `startAnimation`, `stopAnimation`, and `updateAnimation`. The `startAnimation` method takes a single String parameter specifying the name of the animation sequence to run. When the method is called, the current animation sequence is changed and the animation timer is reset to zero. The `stopAnimation` method takes no arguments and sets the
current animation to null. The `updateAnimation` method takes a single floating-point number parameter that specifies how much time has elapsed in the current animation sequence since the previous rendering of the model. When a model consists of multiple sub-meshes, each sub-mesh must also have its animation updated.

```java
// Begin an animation sequence for all submeshes of the model
Iterator<SceneNode> itr = model.getChildren();
while (itr.hasNext()) {
    Model3DTriMesh mesh = ((Model3DTriMesh)itr.next());
    mesh.startAnimation("Walk");
}
```

Figure 4.3 Example of starting an animation sequence

```java
public void update(float time)
{
    // Update the animation time for the model and all submeshes
    Iterator<SceneNode> itr = model.getChildren();
    while (itr.hasNext()) {
        Model3DTriMesh submesh = (Model3DTriMesh) itr.next();
        submesh.updateAnimation(time);
    }
}
```

Figure 4.4 Example of updating a model’s current animation
4.3 Shader Programs

```java
// Create a GLSL shader program from external source files
GLSLShaderProgram shader = new GLSLShaderProgram("basic.vert", "basic.frag");
```

Figure 4.5 Example of creating a GLSL shader program

Creating a GLSLShaderProgram requires file names from which to load the shader source code. The constructor takes two String parameters: the first specifying the file name containing source code for the vertex shader, and the second specifying the file name containing the fragment shader source code. The files `basic.vert` and `basic.frag` are expected to contain valid GLSL program text.

```java
//Attach the shader to all submeshes of the model
Iterator<SceneNode> itr = model.getChildren();
while (itr.hasNext()) {
    Model3DTriMesh submesh = ((Model3DTriMesh)itr.next());
    submesh.setShaderProgram(shader);
}
```

Figure 4.6 Example of attaching a shader program to a hierarchical model

Shader programs are designed to be used with Model3DTriMesh objects. The Model3DTriMesh class provides a method `setShaderProgram` that takes a GLSLShaderProgram object as a parameter. When this method is called, SAGE will render the mesh using the shader program instead of the fixed function pipeline. If a model consists of multiple sub-meshes, each of which is a Model3DTriMesh, `setShaderProgram` must be called on each sub-mesh individually.
Figure 4.7 illustrates how to create uniform variables that can be attached to shader programs. There are multiple subclasses of GLSLUniform, depending on the number of elements and the data-type in the uniform variable. The number after the name specifies the number of elements that variable contains. The last letter in the name refers to the data-type of the variable, where `i` represents integer data and `f` represents floating-point data. The constructor for a GLSLUniform object takes a single String parameter that is the name of the variable as declared in the shader source code. The `setValue` method is the way to specify the value of a GLSLUniform variable. If the `setValue` method is not called explicitly, the variable will use a default value of zero.

```java
// Create a GLSL shader program from external source files
GLSLShaderProgram shader = new GLSLShaderProgram("basic.vert", "normalmap.frag");

// Create uniform variables for the shader to use
GLSLUniform1i colorTexture = new GLSLUniform1i("colorTexture");
colorTexture.setValue(0);

GLSLUniform1i normalMap = new GLSLUniform1i("normalMap");
normalMap.setValue(1);

GLSLUniform4f lightPosition = new GLSLUniform4f("lightPosition");
lightPosition.setValue(new Float[]{1.0f, 5.0f, 0.0f, 0.0f});
```
Chapter 5

FUTURE WORK

5.1 Model Loading

OgreXML is not the most widely used model format in the computer graphics industry. However, it was a good choice to integrate into SAGE due to its easy to understand file structure and it led to the development of animation support. Many third-party utilities can be found for commercial games that are able to extract and export digital content in a variety of file formats. In the future, it would be useful to investigate these tools to see which file format is the most common amongst games of all genres.

If a common file format can be identified, support for it would be a valuable addition to SAGE. It would be easy to obtain high-quality animated models from a multitude of existing popular games via third-party utilities. The goal is for students to be able to focus on the architecture of their game rather than the content. Although, using existing professionally created models will definitely improve the visual quality of the games produced.

5.2 Key-frame Interpolation

The lack of support for smooth interpolation between key-frames greatly affects visual quality if the model has any animation sequences. This will be most apparent if portions of the model move along a spline curve, such as a tail. By decomposing the animation matrices into a translation vector and a rotation quaternion, SAGE would be able to accurately interpolate vertex positions in between key-frames leading to much more realistic animation.
5.3 GPU Skinning

The disadvantage of the current animation framework is that it relies on the computer’s central processing unit for doing all of the transformations. Each vertex and normal must be recalculated through expensive matrix operations, and the buffers for position and normal data need to be refilled. The problem compounds itself when there are multiple animated models inside the viewing frustum. The solution is to provide all the data required to do the animation to the vertex shader and do the calculations on the graphics hardware.

The reason for implementing animation on the CPU in SAGE is twofold: there is a limitation on the number of uniform variables available in each shader stage and it would put too much pressure on the student to write a vertex shader capable of doing the animations. In modern graphics hardware, the number of uniform variables allowed in the vertex shader stage of the pipeline is usually at least 1024 in OpenGL 3.0[22]. While this number appears to be more than sufficient for most applications, the number of uniforms actually refers to individual floating point values. Each matrix passed to the shader occupies 16 uniform locations, quickly limiting the number of joints available for an animated model. Representing a key-frame as a vector and quaternion halves the number of uniforms required to transfer the data.
package sage.animation;

import graphicslib3D.Vector3D;
import java.util.ArrayList;
import java.util.Collection;

/**
 * An AnimationKey-frame contains the data describing a single <I>key-frame</I>
 * of an animation: rotation and translation transforms to be applied to an object (e.g. a Joint),
 * along with a <I>time</i> at which the object should arrive at the specified transformed orientation.
 * @author John Clevenger
 */
public class AnimationKey-frame implements Comparable<AnimationKey-frame>{

    private float time;
    private Matrix3D transform;
    private Joint joint;

    /**
     * This constructor creates a default AnimationKey-frame with a <I>time</i> of zero and both rotation and translations also zero.
     */
    public AnimationKey-frame () {
        this.time = 0;
        transform = new Matrix3D();
        joint = null;
    }

    /**
     * Constructs a AnimationKey-frame that is associated with a specific Joint and time.
     * @param joint The Joint
     * @param time The time
     */
    public AnimationKey-frame(Joint joint, float time) {
        this();
        this.joint = joint;
        this.time = time;
    }
}
/**
 * This method rotates the {@link AnimationKey-frame}'s current
 * transformation around an angle-axis.
 * @param angle The angle to rotate
 * @param rx The X component of the axis
 * @param ry The Y component of the axis
 * @param rz The Z component of the axis
 */
public void rotate(float angle, float rx, float ry, float rz) {
    transform.rotate(angle, new Vector3D(rx, ry, rz));
}

/**
 * This method translates the {@link AnimationKey-frame}'s current
 * transformation.
 * @param tx The translation in the X direction
 * @param ty The translation in the Y direction
 * @param tz The translation in the Z direction
 */
public void translate(float tx, float ty, float tz) {
    transform.translate(tx, ty, tz);
}

/**
 * This method gets the {@link AnimationKey-frame}'s current
 * transformation.
 * @return the current transformation of the key-frame/
 */
public Matrix3D getTransform() {
    return transform;
}

/**
 * This method compares two {@link AnimationKey-frame}s in order to
determine sorting order. {@link AnimationKey-frame}s are sorted by time in
ascending order.
 */
public int compareTo(AnimationKey-frame frame) {
    if(this.time - frame.time > 0)
        return 1;
    else if (this.time - frame.time < 0)
        return -1;
    else
        return 0;
}
* This method will find the nearest {@link AnimationKey-frame} in a
collection * before or equal to the given time.
* @param c The collection of AnimationKey-frames
* @param time The time to find
* @return The nearest key-frame before or equal to the given time.
*/
public static AnimationKey-frame find (Collection<AnimationKey-frame> c, 
float time)
{
    AnimationKey-frame nearest = ((ArrayList<AnimationKey-frame>)c).get(0);
    for (AnimationKey-frame frame : c)
    {
        if (frame.time <= time)
            nearest = frame;
    }
    return nearest;
}
package sage.animation;

import java.util.ArrayList;
import java.util.Collection;
import java.util.TreeMap;

/**
   * An {@link AnimationSequence} holds a sequence of {@link AnimationKey-frame}s
   * defining a particular animation.
   * Each {@link AnimationSequence} has a <I>name</I> (e.g. "Walk", "Run",
   * "Death", "Rest", or something similar) and a set of
   * {@link AnimationKey-frame}s describing the animation implied by the name
   * (walk, run, etc.)
   *
   * @author John Clevenger
   *
   */
public class AnimationSequence {

    private String name;

    private TreeMap<Integer, ArrayList<AnimationKey-frame>> key-frames;

    private float length;

    /**
     * This constructor creates a default {@link AnimationSequence} whose
     * name is the (@link String) "NoName" and which has an empty set of
     * (@link AnimationKey-frame)s.
     */
    public AnimationSequence() {
        this.name = "NoName";
        key-frames = new TreeMap<Integer, ArrayList<AnimationKey-frame>>();
        length = 0;
    }

    /**
     * This constructor creates an {@link AnimationSequence} with the
     * specified name and an empty set of {@link AnimationKey-frame}s.
     *
     * @param name The name of the newly-created {@link AnimationSequence}
     */
    public AnimationSequence(String name) {
        this();
        this.name = name;
    }

    public AnimationSequence(String name, float length) {
        this();
        this.name = name;
        this.length = length;
    }

    /**
     * This method adds the collection of {@link AnimationKey-frame}s to
     * the {@link Joint} specified by ID.
     */
    public void addAnimationKey-frames(Integer boneID, Collection<AnimationKey-frame> c) {
        }

if (key-frames == null)
    key-frames = new TreeMap<Integer, ArrayList<AnimationKey-frame>>();

if (boneID == -1 || c == null)
    throw new RuntimeException("Error: invalid bone ID");

key-frames.put(boneID, new ArrayList<AnimationKey-frame>(c));

/**
 * This method... is currently incomplete.
 * @param time
 * @param repeatType
 * @param speed
 */
public void update(double time, int repeatType, double speed) {
    // TODO Auto-generated method stub
}

/**
 * This method... is currently incomplete.
 * @param time
 */
public void update(double time) {
}

/**
 * This method returns the collection of key-frames associated with
 * the given (@link Joint) ID.
 * @param bone The ID of the Joint
 * @return The collection of key-frames associated with the Joint
 */
public ArrayList<AnimationKey-frame> getKey-frames(int bone) {
    return key-frames.get(bone);
}

/**
 * This method returns the name of the animation sequence
 * @return Returns the name of the animation sequence
 */
public String getName() {
    return name;
}

/**
 * This method returns the length of the animation sequence.
 * @return Returns the length of the animation sequence in seconds.
 */
public float getLength() {
    return length;
}
package sage.animation;

import graphicslib3D.Matrix3D;
import graphicslib3D.Vector3D;
import graphicslib3D.Vertex3D;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import sage.animation.AnimationController;
import sage.animation.AnimationKey-frame;
import sage.animation.AnimationSequence;

/**
 * {@link Joint} represents a joint in the animation skeleton for a model.
 * A {@link Joint} has a <I>name</i>, a <I>parent joint</i> (unless it is the
 * root joint of the model), and potentially one or more <I>child</i> joints. Each (link Joint) also has the following components:

 * <ul>
 * <li> A set of <I>initial transformations</i> which specify the rotation and translation (relative to the parent
 * joint) for the initial (non-animated, "at-rest") orientation of the
 * (@link Joint).
 * <li> A <I>fromParentSpace</i> transformation which transforms the initial ("at-rest") parent joint coordinate system
to the initial ("at-rest") coordinate system of this (link Joint).
 * <li> A <I>fromModelSpace</i> transformation which transforms the model space coordinate system
to the initial ("at-rest") coordinate system of this (link Joint). The <code>fromModelSpace</code> transform is the concatenation of
the <code>fromParentSpace</code> transforms of all the ancestor joints
of this (link Joint). (Note that the <I>inverse</i> of the <code>fromModelSpace</code> transform gives the transform which will convert <I>vertices</i> from
model space to the space of this (link Joint).)
 * <li> An <I>animationFromParentSpace</i> transformation which transforms objects in the parent joint coordinate system
(with all appropriate animation transforms already applied in the
parent) to objects in the the coordinate system of this (link Joint) given that the joint has "moved" due to the current animation
transform.
 * <li> An <I>animationFromModelSpace</i> transformation which transforms objects the model space coordinate system
to the coordinate system of this (link Joint). The <code>animationFromModelSpace</code> transform is the concatenation
of the <code>animationFromParentSpace</code> transforms of all the
ancestor joints of this (link Joint); this is the transform which is
used to animate a vertex attached to this (link Joint) (given that the
vertex has first been transformed to this joint's coordinate space
by the application of the Joint's <I>fromModelSpace</i> transform).
 * <li> An (link AnimationController) which manages the set of
{@link AnimationSequence}s attached to this (link Joint).
 * Note that the (link AnimationController) maybe be null; for example,
 * if the (link Joint) is managing its own (link AnimationKey-frame)s
public class Joint {

private String name;
private ArrayList<Joint> children;
private Matrix3D fromParentSpace;
private Matrix3D fromModelSpace;
private Matrix3D animRelativeToModelSpace;
private Matrix3D animRelativeToParentSpace;
private Joint parent;

/**
 * Creates a Joint with the name "NoName", an empty children list, and Identity (zero) initial transformations.
 */
public Joint() {
    super();
    this.children = new ArrayList<Joint>();
    this.fromParentSpace = new Matrix3D();
    this.fromModelSpace = new Matrix3D();
    this.animRelativeToModelSpace = new Matrix3D();
    this.animRelativeToParentSpace = new Matrix3D();

    parent = null;
}

/**
 * Creates a Joint which has the specified name, an empty children list, and Identity (zero) initial transformations.
 * @param name
 */
public Joint(String name) {
    this();
    this.name = name;
}

/**
 * Creates a Joint which has the specified name,
* an empty children list, and specified initial transformations.

* @param translation Translation in X,Y,Z direction
* @param angle The angle to rotate
* @param axis The axis about which to rotate
*
/**
 public Joint(float[] translation, float angle, float[] axis) {
    this();
    fromParentSpace.translate(translation[0], translation[1], translation[2]);
    fromParentSpace.rotate(angle, new Vector3D(axis[0], axis[1], axis[2]));
}

/**
 * This method sets the Joint's initial translation.
 * @param tx The translation in the X direction
 * @param ty The translation in the Y direction
 * @param tz The translation in the Z direction
 */
public void initialTranslation(float tx, float ty, float tz) {
    this.fromParentSpace.translate(tx, ty, tz);
}

/**
 * This method sets the Joint's initial rotation as an angle-axis.
 * @param angle The angle in which to rotate
 * @param rx The X component of the axis
 * @param ry The Y component of the axis
 * @param rz The Z component of the axis
 */
public void initialRotation(float angle, float rx, float ry, float rz) {
    this.fromParentSpace.rotate(angle, new Vector3D(rx, ry, rz));
}

/**
 * This method sets the parent of this Joint.
 * @param parent The parent
 */
private void setParent(Joint parent) {
    this.parent = parent;
}

/**
 * This method checks to see whether this Joint has a parent.
 * @return True if this Joint has a parent, false otherwise
 */
public boolean hasParentJoint() {
    return parent != null;
}
/**
 * This method gets the parent (@link Joint) of this Joint.
 * @return The parent Joint.
 */
public Joint getParent() {
    return parent;
}

/**
 * Returns a String giving the name of this (@link Joint).
 * Note that Strings are immutable and there is no <code>setName()</code>
 * method in this class; this is to insure that the (@link Joint)s name
 * cannot be made inconsistent with the name in the external model file
 * from which this (@link Joint) was constructed.
 */
public String getName() {
    return name;
}

/**
 * This method gets the transformation of the current animation
 * relative to model space.
 * @return The animation to model space transformation
 */
public Matrix3D getAnimRelativeToModelSpace() {
    return animRelativeToModelSpace;
}

/**
 * This method sets the transformation of the current animation
 * relative to model space.
 * @param animRelativeToModelSpace The animation to model space
 * transformation
 */
public void setAnimRelativeToModelSpace(Matrix3D animRelativeToModelSpace) {
    this.animRelativeToModelSpace = animRelativeToModelSpace;
}

/**
 * This method gets the transformation of the current animation
 * relative to parent space.
 * @return The animation to parent space transformation
 */
public Matrix3D getAnimRelativeToParentSpace() {
    return animRelativeToParentSpace;
}

/**
 * This method sets the transformation of the current animation
 * relative to parent space.
 */
* @param animRelativeToParentSpace The animation to parent space transformation
 */
public void setAnimRelativeToParentSpace(Matrix3D animRelativeToParentSpace) {
  this.animRelativeToParentSpace = animRelativeToParentSpace;
}

/**
 * This method gets the transformation from parent to Joint space.
 * @return The Joint to parent space transformation
 */
public Matrix3D getFromParentSpace() {
  return fromParentSpace;
}

/**
 * This method sets the transformation from parent to Joint space.
 * @param fromParentSpace the parent to Joint space transformation
 */
public void setFromParentSpace(Matrix3D fromParentSpace) {
  this.fromParentSpace = fromParentSpace;
}

/**
 * This method gets the transformation from model to Joint space.
 * @return The Joint to model space transformation
 */
public Matrix3D getFromModelSpace() {
  return fromModelSpace;
}

/**
 * This method sets the transformation from model to Joint space.
 * @param fromModelSpace the model to Joint space transformation
 */
public void setFromModelSpace(Matrix3D fromModelSpace) {
  this.fromModelSpace = fromModelSpace;
}

/**
 * Adds the specified Joint as a child of this Joint. This method throws an exception if the specified Joint is null or if it is not a Joint.
 * @param j The joint to be added.
 */
public void addChild(Joint j) {
  if (j == null) {
    throw new NullPointerException();
  }

  if (! (j instanceof Joint)) {
    throw new IllegalArgumentException("Attempt to add an object" +

"which is not a Joint as a child of a Joint";
}
if (children == null) {
    children = new ArrayList&lt;Joint&gt;();
}
children.add(j);
j.setParent(this);
}
/**
 * Returns an iterator for the children of this {@link Joint}.
 */
public Iterator&lt;Joint&gt; getChildren() {
    return children.iterator();
}

public String toString()
{
    return name;//return this.fromParentSpace.toString();
}
/**
 * This method will initialize a hierarchy of {@link Joint}s to compute
 * their fromModelSpace transformations.
 *
 * @param c The collection of Joints
 */
public static void initialize(Collection&lt;Joint&gt; c)
{
    for (Joint joint : c)
    {
        //initialize the matrix that moves the world (model) axes all the
        //way to this joint's axes
        if (!joint.hasParentJoint())
        {
            //there is no parent joint; the from-Model-Space transform is
            just the local transform
            Matrix3D mat1 = new Matrix3D(joint.fromParentSpace.getValues());
            joint.fromModelSpace = mat1;
        }
        else
        {
            Joint parentJoint = joint.parent;
            Matrix3D mat2 = new Matrix3D(parentJoint.fromModelSpace.getValues());
            mat2.concatenate(joint.fromParentSpace);
            joint.fromModelSpace = mat2;
        }
    }
}
package sage.model.loader;

import java.nio.FloatBuffer;
import java.nio.IntBuffer;
import java.util.ArrayList;
import java.util.Stack;
import java.util.TreeMap;
import java.util.Vector;
import javax.xml.namespace.QName;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
import com.jogamp.common.nio.Buffers;

/**
 * {@link SAGEXMLParser} is an abstract framework for parsing
 * XML files of three-dimensional models. It contains generic methods
 * for extracting XML element names and values along with methods
 * handling converting collections of Objects to arrays of primitives.
 * 
 * @author Morgan Darke
 */

public abstract class SAGEXMLParser {

    protected Stack<XMLEvent> stack;

    /**
     * Creates a {@link SAGEXMLParser} with an empty stack.
     */
    protected SAGEXMLParser() {
        stack = new Stack<XMLEvent>();
    }

    /**
     * This method gets the name of the XML element.
     * 
     * @param event The XML event that occurred.
     * @return The name of the XML element
     */
    protected String getElementName(XMLEvent event) {
        if (event.isStartElement())
            return event.asStartElement().getName().getLocalPart();
        else
            return event.asEndElement().getName().getLocalPart();
    }

    /**
     * This method gets the value of the XML attribute as a String.
     */

protected String getAttributeValue(StartElement element, String name) throws Exception {
    Attribute attribute = element.getAttributeByName(new QName(name));
    if (attribute != null)
        return attribute.getValue();
    else
        throw new XMLParserException("Error: Can not find any attribute named " + name, element);
}

protected Float getAttributeValueAsFloat(StartElement element, String name) throws Exception {
    Attribute attribute = element.getAttributeByName(new QName(name));
    if (attribute != null)
        return Float.valueOf(attribute.getValue());
    else
        throw new XMLParserException("Error: Can not find any attribute named " + name, element);
}

protected Integer getAttributeValueAsInteger(StartElement element, String name) throws Exception {
    Attribute attribute = element.getAttributeByName(new QName(name));
    if (attribute != null)
        return Integer.valueOf(attribute.getValue());
    else
        throw new XMLParserException("Error: Can not find any attribute named " + name, element);
}
/**
 * This method checks to see if two XML elements are equal.
 * @param element1 The first XML Element
 * @param element2 The second XML Element
 * @return true if and only if neither element is null and
 * both of the elements have the same name
 */
protected boolean isMatchingElements(StartElement element1, EndElement element2) {
    return (element1 != null && element2 != null) &&
            element1.getName().equals(element2.getName());
}

/**
 * This method pops the element on top of the stack and checks
 * that it matches with the current EndElement.
 * @param event The XML EndElement
 * @throws XMLParserException If the two elements did not match, meaning
 * the XML data is malformed.
 */
protected void pop(XMLEvent event) throws XMLParserException {
    if (stack.isEmpty() || event == null)
        throw new XMLParserException("Error: Mismatching number of start
and end elements ", event);
    StartElement previous = stack.pop().asStartElement();
    if (!getElementName(previous).equalsIgnoreCase(getElementName(event)))
        throw new XMLParserException("Error: Mismatching number of start
and end elements ", event);
}

/**
 * This method pushes an XMLEvent onto the top of the stacks.
 * @param event The XML event
 */
protected void push(XMLEvent event) {
    stack.push(event);
}

/**
 * This method converts a collection of Integer objects into
 * an array of primitive types.
 * @param array The collection of Integer objects.
 * @return an array of float primitives.
 */
protected int[] toIntegerPrimitiveArray(Integer[] array) {
int[] primitive = new int[array.length];

for (int i = 0; i < primitive.length; i++)
    primitive[i] = array[i];

return primitive;
}

/**
 * This method converts a collection of {@link Integer}s into
 * an array of primitive types.
 * @param vector The collection of Integer objects.
 * @return an array of float primitives.
 */
protected int[] toIntegerPrimitiveArray(Vector<Integer> vector) {
    int[] primitive = new int[vector.size()];

    for (int i = 0; i < primitive.length; i++)
        primitive[i] = vector.get(i);

    return primitive;
}

/**
 * This method converts a collection of {@link Float}s into
 * an array of primitive types.
 * @param array The collection of Float objects.
 * @return an array of float primitives.
 */
protected float[] toFloatPrimitiveArray(Float[] array) {
    float[] primitive = new float[array.length];

    for (int i = 0; i < primitive.length; i++)
        primitive[i] = array[i];

    return primitive;
}

/**
 * This method converts a collection of {@link Float}s into
 * an array of primitive types.
 * @param vector The collection of Float objects.
 * @return an array of float primitives.
 */
protected float[] toFloatPrimitiveArray(Vector<Float> vector) {
    float[] primitive = new float[vector.size()];

    for (int i = 0; i < primitive.length; i++)
        primitive[i] = vector.get(i);
```java
return primitive;
}

/**
 * This method constructs an `{@link FloatBuffer}` with the contents
 * of a collection of `{@link Float}s.
 * @param vector The Vector of Floats.
 * @return A FloatBuffer containing the values in the collection.
 */
protected FloatBuffer toFloatBuffer(Vector<Float> vector) {
    return Buffers.newDirectFloatBuffer(toFloatPrimitiveArray(vector));
}

/**
 * This method constructs an `{@link IntBuffer}` with the contents
 * of a collection of `{@link Integer}s.
 * @param vector The Vector of Integers.
 * @return An IntBuffer containing the values in the collection.
 */
protected IntBuffer toItemBuffer(Vector<Integer> vector) {
    return Buffers.newDirectIntBuffer(toIntegerPrimitiveArray(vector));
}

/**
 * This method converts a map of key-value pairs into a
 * collection populated by the values.
 * @param map The map of key-value pairs.
 * @return An Array list of only the values inside the map.
 */
protected <K, V> ArrayList<V> mapToArray(TreeMap<K, V> map) {
    ArrayList<V> collection = new ArrayList<V>();
    for (K key : map.keySet())
        collection.add(map.get(key));
    return collection;
}
```
package sage.model.loader;

import javax.xml.stream.events.XMLEvent;

/**
 * {@link XMLParserException} represents an error that occurred during the
 * parsing of an XML file. A {@link XMLParserException} contains an error
 * message, the XML element that caused the exception, and the line number
 * of the file where the error originated.
 * @author Morgan Darke
 * */
 public class XMLParserException extends Exception {

 private static final long serialVersionUID = -6022084825701336179L;

 /**
 * Creates a {@link XMLParserException} with a specified error message,
 * created by the given XMLEvent.
 * */
 public XMLParserException(String message, XMLEvent event) {
 super(message + " - " + event.toString() + " on line " +
 event.getLocation().getLineNumber() + ".");
 }
}
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.Arrays;
import java.util.Iterator;
import java.util.TreeMap;
import java.util.Vector;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
import sage.animation.AnimationSequence;
import sage.animation.Joint;
import sage.display.DisplaySystem;
import sage.model.loader.SAGEXMLParser;
import sage.model.loader.XMLParserException;
import sage.renderer.IRenderer;
import sage.scene.Group;
import sage.scene.Model3DTriMesh;
import sage.scene.SceneNode;
import sage.scene.state.TextureState;
import sage.scene.state.RenderState.RenderStateType;
import sage.texture.Texture;
import sage.texture.TextureManager;

/**
 *{@link OgreXMLParser} is used to parse OgreXML files containing geometry, material, and animation data and turn them into a {@link Group} of {@link Model3DTriMesh}es to be rendered.
 *{@author Morgan Darke}
 */

public class OgreXMLParser extends SAGEXMLParser {

    private static final int MAX_BONES_PER_VERTEX = 4;

    private int vertexCount;
    private Vector<Float> vertices;
    private Vector<Float> normals;
    private Vector<Float> texcoords;
    private Vector<Integer> indices;
    private int[] boneIDs;
    private float[] weights;
    private TreeMap<Integer, Vector<OgreXMLVertexBoneAssignment>> boneAssignments;

private TreeMap<String, AnimationSequence> animations;

private String mesh;

private String materials;

private String animation;

private Group root;

/**
 * Creates a new (OgreXMLParser) which does not
 * have any geometry, material, or animation data.
 */
public OgreXMLParser() {
    mesh = null;
    materials = null;
    animation = null;

    vertices = null;
    normals = null;
    texcoords = null;
    indices = null;
    boneAssignments = null;
    root = null;
}

/**
 * Initializes the collections used by the parser.
 */
private void init() {
    vertices = new Vector<Float>();
    normals = new Vector<Float>();
    texcoords = new Vector<Float>();
    indices = new Vector<Integer>();
    boneAssignments = new TreeMap<Integer, Vector<OgreXMLVertexBoneAssignment>>();
    root = new Group();
}

/**
 * This method loads an OgreXML model and returns a
 * (Group) of (Model3DTriMesh)es.
 * @param meshData The name of the file containing geometry data.
 * @param materialData The name of the file containing material data.
 * @param animationData The name of the file containing animation data.
 * @return a (Group) of (Model3DTriMesh) representing the
 * model.
 * @throws Exception On any error during parsing.
 */
public Group loadModel(String meshData, String materialData, String animationData) throws Exception {
    mesh = meshData;
}
materials = materialData;
animation = animationData;

init();

XMLInputFactory inputFactory = XMLInputFactory.newInstance();
InputStream in = new FileInputStream(mesh);
XMLEventReader reader = inputFactory.createXMLEventReader(in);

boolean finished = false;

while (!finished && reader.hasNext()) {
    XMLEvent event = reader.nextEvent();
    if (event.isStartElement()) {
        StartElement element = event.asStartElement();
        push(element);
        if (getElementName(event).equalsIgnoreCase("boneassignments"))
            processBones(reader, element);
        else if (getElementName(event).equalsIgnoreCase("skeletonlink"))
            processAnimation(reader, element);
        else if (getElementName(event).equalsIgnoreCase("sub-meshes"))
            processSub-meshes(reader, element);
        else if (getElementName(event).equalsIgnoreCase("sharedgeometry"))
            processGeometry(reader, element);
        else if (!getElementName(event).equalsIgnoreCase("mesh"))
            throw new XMLParserException("Error: Unexpected start element ", event);
    } else if (event.isEndElement()) {
        pop(event);
        if (getElementName(event).equalsIgnoreCase("mesh") &&
            stack.isEmpty()) {
            assignWeights();
            // Currently every group in the mesh has its own copy of
            the same geometry

            Iterator<SceneNode> iterator = root.getChildren();
            while (iterator.hasNext()) {
                Model3DTriMesh child = (Model3DTriMesh)
                iterator.next();
                child.setVertexBoneIDs(boneIDs);
                child.setVertexBoneWeights(weights);
                child.setAnimations(animations);
            }
            finished = true;
        } else

            finish();
    }
}

throw new XMLParserException("Error: Unexpected end of file ", event);
    }
}
if (!finished)
    throw new Exception("Error: Unexpected end of file");

return root;

/**
 * This method begins to parse the geometry data.
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processGeometry(XMLEventReader reader, StartElement currentElement) throws Exception {
    StartElement element = null;
    boolean finished = false;

    vertexCount = getAttributeValueAsInteger(currentElement, "vertexcount");

    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();

        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);

            if (getElementName(event).equalsIgnoreCase("vertexbuffer"))
                processVertexBuffer(reader, element);
            else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {

            int numVertices = vertices.size() / 3;

            if (numVertices != vertexCount)
                throw new XMLParserException("Error: Vertex buffer size (" + numVertices + ") does not match vertex count (" + vertexCount + ").", event);

            pop(event);
            if (getElementName(event).equalsIgnoreCase("sharedgeometry")) {
                int numVertices = vertices.size() / 3;
                if (numVertices != vertexCount)
                    throw new XMLParserException("Error: Vertex buffer size (" + numVertices + ") does not match vertex count (" + vertexCount + ").", event);

            }
        }
    }
}
finished = true;

} */

private void processVertexBuffer(XMLEventReader reader, StartElement currentElement)
throws Exception {

StartElement element = null;
boolean finished = false;

while (!finished && reader.hasNext()) {
    XMLEvent event = reader.nextEvent();

    if (event.isStartElement()) {
        element = event.asStartElement();
        push(element);

        if (getElementName(event).equalsIgnoreCase("vertex"))
            processVertex(reader, element);
        else
            throw new XMLParserException("Error: Unexpected start element ", event);
    }

    else if (event.isEndElement()) {
        pop(event);

        if (getElementName(event).equalsIgnoreCase("vertexbuffer"))
            finished = true;
    }
}

} /* This method processes a single vertex which contains
* position coordinates, texture coordinates, and/or normals.
* @param reader The XMLReader for the skeleton file.
* @param currentElement The current XML StartElement
* @throws Exception During any errors in the parsing process.
*/

private void processVertex(XMLEventReader reader, StartElement currentElement) throws Exception {

}
StartElement element = null;
boolean finished = false;

while (!finished && reader.hasNext()) {
    XMLEvent event = reader.nextEvent();
    if (event.isStartElement()) {
        element = event.asStartElement();
        push(element);
        if (getElementName(event).equalsIgnoreCase("position")) {
            vertices.add(getAttributeValueAsFloat(element, "x"));
            vertices.add(getAttributeValueAsFloat(element, "y"));
            vertices.add(getAttributeValueAsFloat(element, "z"));
        } else if (getElementName(event).equalsIgnoreCase("normal")) {
            normals.add(getAttributeValueAsFloat(element, "x"));
            normals.add(getAttributeValueAsFloat(element, "y"));
            normals.add(getAttributeValueAsFloat(element, "z"));
        } else if (getElementName(event).equalsIgnoreCase("texcoord")) {
            texcoords.add(getAttributeValueAsFloat(element, "u"));
            texcoords.add(getAttributeValueAsFloat(element, "v"));
        } else {
            throw new XMLParserException("Error: Unexpected start element ", event);
        }
    } else if (event.isEndElement()) {
        pop(event);
        if (getElementName(event).equalsIgnoreCase("vertex")) {
            finished = true;
        }
    }
}

/**
* This method begins to process the sub-meshes of the model.
* @param reader The XMLReader for the skeleton file.
* @param currentElement The current XML StartElement
* @throws Exception During any errors in the parsing process.
*/
private void processSub-meshes(XMLEventReader reader, StartElement currentElement) throws Exception {
    StartElement element = null;
    boolean finished = false;
    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
    }
}
if (event.isStartElement()) {
  element = event.asStartElement();
  push(element);
  if (getElementName(event).equalsIgnoreCase("submesh"))
    processSubmesh(reader, element);
  else
    throw new XMLParserException("Error: Unexpected start element ", event);
}
else if (event.isEndElement()) {
  pop(event);
  if (getElementName(event).equalsIgnoreCase("sub-meshes"))
    finished = true;
}
}

/**
 * This method processes a single submesh within the group.
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processSubmesh(XMLEventReader reader, StartElement currentElement) throws Exception {
  Model3DTriMesh submesh = new Model3DTriMesh();

  OgreXMLMaterialParser mat = new OgreXMLMaterialParser(materials);

  submesh.setTextureFilename( mat.getAttributeValue( getAttributeValue(currentElement, "material")));

  StartElement element = null;
  boolean finished = false;

  while (!finished && reader.hasNext()) {
    XMLEvent event = reader.nextEvent();

    if (event.isStartElement()) {
      element = event.asStartElement();
      push(element);
      if (getElementName(event).equalsIgnoreCase("faces"))
        processFaces(reader, element);
      else
        throw new XMLParserException("Error: Unexpected start element ", event);
    }
  }
}
else if (event.isEndElement()) {
    pop(event);
    if (getElementName(event).equalsIgnoreCase("submesh")) {
        submesh.setVertexBuffer(toFloatBuffer(vertices));
        submesh.setNormalBuffer(toFloatBuffer(normals));
        submesh.setTextureBuffer(toFloatBuffer(texcoords));
        submesh.setIndexBuffer(toIntBuffer(indices));

        //Added code:
        //create a Texture object from the texture image file (if there is one)
        String texFileName = submesh.getTextureFileName();
        if (texFileName != null) {
            //create a texture object from the texture file
            Texture tex = TextureManager.loadTexture2D(texFileName);

            //construct a TextureState describing this texture
            IRenderer renderer = DisplaySystem.getCurrentDisplaySystem().getRenderer();
            TextureState texState = (TextureState) renderer.createRenderState(RenderStateType.Texture);
            texState.setTexture(tex);
            texState.setEnabled(true);

            //install the TextureState into the submesh
            submesh.setRenderState(texState);
        }
    }
}

root.addChild(submesh);
indices.clear();
finished = true;
}
XMLEvent event = reader.nextEvent();

if (event.isStartElement()) {
    element = event.asStartElement();
    push(element);
    if (getElementName(event).equalsIgnoreCase("face")) {
        indices.add(getAttributeValueAsInteger(element, "v1"));
        indices.add(getAttributeValueAsInteger(element, "v2"));
        indices.add(getAttributeValueAsInteger(element, "v3"));
    } else 
        throw new XMLParserException("Error: Unexpected start element ", event);
} else if (event.isEndElement()) {
    pop(event);
    if (getElementName(event).equalsIgnoreCase("faces"))
        finished = true;
}

/**
 * This method will process animation data by creating
 * a {@link OgreXMLAnimationParser}
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processAnimation(XMLEventReader reader, StartElement currentElement) throws Exception {
    boolean finished = false;

    OgreXMLAnimationParser animationParser = new OgreXMLAnimationParser(root);
    animations = animationParser.process(animation);

    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            pop(event);
        }
    }
}
if (getElementName(event).equalsIgnoreCase("skeletonlink"))
    finished = true;
}
}

/**
 * This method will process bone assignments describing which
 * @link Joint)s are associated with each vertex and the
 * how much the vertex is weighted by that @link Joint).
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processBones(XMLEventReader reader,StartElement currentElement) throws Exception {
    StartElement element = null;
    boolean finished = false;
    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("vertexboneassignment")) {
                int vertexindex = getAttributeValueAsInteger(element, "vertexindex");
                int boneindex = getAttributeValueAsInteger(element, "boneindex");
                float weight = getAttributeValueAsFloat(element, "weight");
                Vector<OgreXMLVertexBoneAssignment> list = boneAssignments.get(vertexindex);
                if (list == null) {
                    list = new Vector<OgreXMLVertexBoneAssignment>();
                    list.add(new OgreXMLVertexBoneAssignment(boneindex, weight));
                    boneAssignments.put(vertexindex, list);
                } else
                    list.add(new OgreXMLVertexBoneAssignment(boneindex, weight));
            } else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            
        }
    }
}
pop(event);

if (getElementName(event).equalsIgnoreCase("boneassignments"))
    finished = true;
}
}
/**
 * This function fills arrays of bone IDs and weights
 * for each vertex. Each vertex has up to MAX_BONES_PER_VERTEX
 * bones associated with it. Slots for bones which are not used
 * will have IDs and weights of zero and will not contribute.
 * @throws Exception if any bone weights are invalid.
 */
private void assignWeights() throws Exception {
    weights = new float[MAX_BONES_PER_VERTEX * vertexCount];
    boneIDs = new int[MAX_BONES_PER_VERTEX * vertexCount];

    Arrays.fill(weights, 0);
    Arrays.fill(boneIDs, 0);

    // Each key corresponds to one vertex
    for (Integer key : boneAssignments.keySet()) {
        // Each vertex can be associates with up to MAX_BONES_PER_VERTEX bones
        Vector<OgreXMLVertexBoneAssignment> assignments =
            boneAssignments.get(key);

        int count = 0;

        for (OgreXMLVertexBoneAssignment assignment : assignments) {

            if (count >= MAX_BONES_PER_VERTEX)
                throw new Exception("Error: Only " + MAX_BONES_PER_VERTEX +
" bones per vertex is currently supported.");

            int index = (MAX_BONES_PER_VERTEX * key) + count;

            boneIDs[index] = assignment.getBoneID();
            weights[index] = assignment.getWeight();

            if (weights[index] > 1.0f || weights[index] < 0.0f)
                throw new RuntimeException("Bone weights must be between
0.0 and 1.0.");

            count++;
        }
    }
}
package sage.model.loader.ogreXML;

import java.io.FileInputStream;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeMap;
import java.util.TreeSet;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
import sage.animation.AnimationKeyframe;
import sage.animation.AnimationSequence;
import sage.animation.Joint;
import sage.model.loader.SAGEXMLParser;
import sage.model.loader.XMLError;
import sage.scene.Group;
import sage.scene.Model3DTriMesh;
import sage.scene.SceneNode;

/**
 * OgreXMLAnimationParser parses animation data found in an OgreXML.skeleton file
 * and populates data in a hierarchy of Model3DTriMeshes. The OgreXML
 * skeleton file structure consists of bones or Joints in their initial
 * poses, a hierarchy describing the parent to child relationships between the Joints, and zero or
 * more animation sequences containing AnimationKey-frame data.
 *
 * @author Morgan Darke
 *
 */

public class OgreXMLAnimationParser extends SAGEXMLParser {

    private TreeMap<Integer, Joint> joints; // A map of Vertex IDs to Joints
    private TreeMap<String, AnimationSequence> animations; // A map of
    private AnimationSequence currentAnimation;
    private Joint currentJoint;
    private int currentTrackID;
    private Group root;

    /**
     * Creates an OgreXMLAnimationParser that will populate the given
     * group of Model3DTriMeshes with animation data.
     */
OgreXMLAnimationParser(Group root) {

    joints = new TreeMap<Integer, Joint>();
    animations = new TreeMap<String, AnimationSequence>();
    currentAnimation = null;
    currentJoint = null;
    currentTrackID = -1;
    this.root = root;
}

/**
 * This is the top-level method that begins the recursive parsing of the
 * animation data.
 * @param URL The URL from which to read the animation data.
 * @return Returns a mapping of animation names to animation sequences.
 * @throws Exception During any errors in the parsing process.
 */
public TreeMap<String, AnimationSequence> process(String URL) throws Exception {

    XMLInputFactory inputFactory = XMLInputFactory.newInstance();
    InputStream in = new FileInputStream(URL);
    XMLEventReader reader = inputFactory.createXMLEventReader(in);

    boolean finished = false;

    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()){
            StartElement element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("bones"))
                process Bones(reader, element);
            else if (getElementName(event).equalsIgnoreCase("bonehierarchy"))
                processBoneHierarchy(reader, element);
            else if (getElementName(event).equalsIgnoreCase("animations"))
                processAnimations(reader, element);
            else if (!getElementName(event).equalsIgnoreCase("skeleton"))
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            pop(event);
            if (getElementName(event).equalsIgnoreCase("skeleton") &&
                stack.isEmpty())
                finished = true;
            else
                throw new XMLParserException("Error: Unexpected end of file ", event);
        }
    }
}
return animations;

private void processBones(XMLEventReader reader, StartElement currentElement) throws Exception {
   StartElement element = null;
    boolean finished = false;
    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("bone"))
                processBone(reader, element);
            else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            pop(event);
            if (getElementName(event).equalsIgnoreCase("bones"))
                finished = true;
        }
    }
}

private void processBone(XMLEventReader reader, StartElement currentElement) throws Exception {
    boolean finished = false;
    int ID = getAttributeValueAsInteger(currentElement, "id");
    Joint joint = new Joint(String.valueOf(ID));
float angle = 0;
while (!finished && reader.hasNext())
{
    XMLEvent event = reader.nextEvent();
    if (event.isStartElement())
    {
        StartElement element = event.asStartElement();
        push(element);
        if (getElementName(event).equalsIgnoreCase("position"))
        {
            float tx = getAttributeValueAsFloat(element, "x");
            float ty = getAttributeValueAsFloat(element, "y");
            float tz = getAttributeValueAsFloat(element, "z");
            joint.initialTranslation(tx, ty, tz);
        }
        else if (getElementName(event).equalsIgnoreCase("rotation"))
        {
            angle = getAttributeValueAsFloat(element, "angle");
        }
        else if (getElementName(event).equalsIgnoreCase("axis"))
        {
            float rx = getAttributeValueAsFloat(element, "x");
            float ry = getAttributeValueAsFloat(element, "y");
            float rz = getAttributeValueAsFloat(element, "z");
            joint.initialRotation(angle, rx, ry, rz);
        }
        else
        {
            throw new XMLParserException("Error: unexpected start element ", event);
        }
    }
    else if (event.isEndElement())
    {
        pop(event);
        if (getElementName(event).equalsIgnoreCase("bone"))
        {
            joints.put(ID, joint);
            finished = true;
        }
    }
}

/**
 * This method processes the {@link Joint} hierarchy. Each {@link Joint}
 * has a parent. Once the complete hierarchy is processed, their dependent
 * transformations are applied to get their configuration in Model space.
 *
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processBoneHierarchy(XMLEventReader reader, StartElement currentElement) throws Exception {
    StartElement element = null;
    boolean finished = false;
    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("boneparent")) {
                int boneID = getAttributeValueAsInteger(element, "bone");
                int parentID = getAttributeValueAsInteger(element, "parent");
                Joint parent = joints.get(parentID);
                if (parent == null)
                    throw new XMLParserException("Error: Parent joint does not exist.", element);
                parent.addChild(joints.get(boneID));
            } else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            pop(event);
            if (getElementName(event).equalsIgnoreCase("bonehierarchy")) {
                ArrayList<Joint> jointArray = mapToArray(joints);
                Joint.initialize(jointArray);
                Iterator<SceneNode> iterator = root.getChildren();
                while (iterator.hasNext()) {
                    Model3DTriMesh child = (Model3DTriMesh)
                        iterator.next();
                    child.setJoints(jointArray);
                }
                finished = true;
            }
        }
    }
}
* This method begins the processing of animation sequences.
* 
* @param reader The XMLReader for the skeleton file.
* @param currentElement The current XML StartElement
* @throws Exception During any errors in the parsing process.
*/
private void processAnimations(XMLEventReader reader, StartElement currentElement) throws Exception {
    StartElement element = null;
    boolean finished = false;
    
    while (!finished || reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("animation"))
                processAnimation(reader, element);
            else
                throw new XMLParserException("Error: Unexpected start element ", event);
        }
        else if (event.isEndElement()) {
            pop(event);
            if (getElementName(event).equalsIgnoreCase("animations"))
                finished = true;
        }
    }
    
    /**
    * This method processes a single animation sequence which contains
    * a <I>name</i> and a <I>length</i>.
    * 
    * @param reader The XMLReader for the skeleton file.
    * @param currentElement The current XML StartElement
    * @throws Exception During any errors in the parsing process.
    */
    private void processAnimation(XMLEventReader reader, StartElement currentElement) throws Exception {
        StartElement element = null;
        boolean finished = false;
        
        String name = getAttributeValue(currentElement, "name");
        float length = getAttributeValueAsFloat(currentElement, "length");
        
        currentAnimation = new AnimationSequence(name, length);
        
        while (!finished || reader.hasNext()) {
            XMLEvent event = reader.nextEvent();
    
        }
if (event.isStartElement()) {
    element = event.asStartElement();
    push(element);
    if (getElementName(event).equalsIgnoreCase("tracks"))
        processTracks(reader, element);
    else
        throw new XMLParserException("Error: Unexpected start element ", event);
} else if (event.isEndElement()) {
    pop(event);
    if (getElementName(event).equalsIgnoreCase("animation")) {
        animations.put(name, currentAnimation);
        currentAnimation = null;
        finished = true;
    }
}
}

/**
 * This method begins to process a single track of an animation sequence.
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processTracks(XMLEventReader reader, StartElement currentElement) throws Exception {
    StartElement element = null;
    boolean finished = false;
    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("track")
                processTrack(reader, element);
            else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            pop(event);
            if (getElementName(event).equalsIgnoreCase("tracks"))
                ...
finished = true;

}  
}

/**
 * This method processes an animation track which contains all the key-frame
 * data for a single Joint.
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @throws Exception During any errors in the parsing process.
 */
private void processTrack(XMLEventReader reader, StartElement currentElement) throws Exception {
    TreeSet<AnimationKey-frame> track = null;
    StartElement element = null;
    boolean finished = false;

    currentTrackID = getAttributeValueAsInteger(currentElement, "bone");
    currentJoint = joints.get(currentTrackID);

    if (currentJoint == null)
        throw new XMLParserException("Error: Animation track for invalid bone ID", currentElement);

    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();

        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);

            if (getElementName(event).equalsIgnoreCase("key-frames"))
                track = processKey-frames(reader, element);
            else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement())
            pop(event);

        if (getElementName(event).equalsIgnoreCase("track"))
            if (track == null || track.size() == 0) {
                // Add an empty animation sequence if a track is present for a joint but no key-frame data declared
                track = new TreeSet<AnimationKey-frame>();
                AnimationKey-frame key-frame = new AnimationKey-frame(currentJoint, 0);
                track.add(key-frame);
            }
        }
    }
//throw new OgreXMLException("Error: Expecting animation tracks, but found none for bone " + currentTrackID, event);
}
currentAnimation.addAnimationKey-frames(currentTrackID, track);
currentTrackID = -1;
currentJoint = null;
finished = true;
}
}

/**
 * This method processes all the key-frames for a specific {@link Joint}.
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @return Returns a TreeSet of{@link AnimationKey-frame}s for all key-frames for a specific {@link Joint}.
 * @throws Exception During any errors in the parsing process.
 */
private TreeSet<AnimationKey-frame> processKey-frames(XMLEventReader reader, StartElement currentElement) throws Exception {
    TreeSet<AnimationKey-frame> key-frames = new TreeSet<AnimationKey-frame>();
    StartElement element = null;
    boolean finished = false;
    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();
        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);
            if (getElementName(event).equalsIgnoreCase("key-frame")) {
                AnimationKey-frame key-frame = processKey-frame(reader, element);
                if (!key-frames.contains(key-frame))
                    key-frames.add(key-frame);
            } else
                throw new XMLParserException("Error: Unexpected start element ", event);
        } else if (event.isEndElement()) {
            pop(event);
        }
    }
}
if (getElementName(event).equalsIgnoreCase("key-frames"))
    finished = true;
}
}
return key-frames;

/**
 * This method processes a @link AnimationKey-frame which describes
 * translation and rotation information for a particular @link Joint
 * at a specified point in time.
 * @param reader The XMLReader for the skeleton file.
 * @param currentElement The current XML StartElement
 * @return Returns a single @link AnimationKey-frame.
 * @throws Exception During any errors in the parsing process.
 */
private AnimationKey-frame processKey-frame(XMLEventReader reader, StartElement currentElement) throws Exception {
   StartElement element = null;
    boolean finished = false;

    float time = getAttributeValueAsFloat(currentElement, "time");
    AnimationKey-frame key-frame = new AnimationKey-frame(currentJoint, time);

    float tx = 0.0f;
    float ty = 0.0f;
    float tz = 0.0f;
    float angle = 0.0f;
    float axis_x = 0.0f;
    float axis_y = 0.0f;
    float axis_z = 0.0f;

    while (!finished && reader.hasNext()) {
        XMLEvent event = reader.nextEvent();

        if (event.isStartElement()) {
            element = event.asStartElement();
            push(element);

            if (getElementName(event).equalsIgnoreCase("translate")) {
                tx = getAttributeValueAsFloat(element, "x");
                ty = getAttributeValueAsFloat(element, "y");
                tz = getAttributeValueAsFloat(element, "z");

                key-frame.translate(tx, ty, tz);
            } else if (getElementName(event).equalsIgnoreCase("rotate"))
                angle = (float) Math.toDegrees(getAttributeValueAsFloat(element, "angle"));
            else if (getElementName(event).equalsIgnoreCase("axis")) {
                axis_x = getAttributeValueAsFloat(element, "x");
            } else if (getElementName(event).equalsIgnoreCase("translate"))
                tx = getAttributeValueAsFloat(element, "x");
        } else if (event.isEndElement()) {
            pop(element);
        }
    }
}
axis_y = getAttributeValueAsFloat(element, "y");
axis_z = getAttributeValueAsFloat(element, "z");

key-frame.rotate(angle, axis_x, axis_y, axis_z);
}
else
    throw new XMLParserException("Error: Unexpected start element ", event);
}
else if (event.isEndElement()) {
    pop(event);
    if (getElementName(event).equalsIgnoreCase("key-frame"))
        finished = true;
}

return key-frame;
package sage.model.loader.ogreXML;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.Stack;

import sage.model.loader.XMLParserException;

/**
 * @link OgreXMLMaterialParser} is used to parse OgreXML .material files to associate textures with sub-meshes.
 * @author Morgan Darke
 */
public class OgreXMLMaterialParser {

    private static final int MATERIAL_INDEX = 1;
    private static final String WHITESPACE = "\s+";
    private File filename;

    /**
     * Creates a (@link OgreXMLMaterialParser) to read from a given File.
     * @param materialFile The material file
     * @throws XMLParserException If the material file can not be found.
     */
    OgreXMLMaterialParser(String materialFile) throws XMLParserException {
        filename = new File(materialFile);
        if (!filename.exists())
            throw new XMLParserException("Error: Unable to find material file " + materialFile, null);
    }

    /**
     * This method extracts the filename of the texture used for the given submesh.
     * @param material The name of the material
     * @return A String of the file name for the image used.
     * @throws Exception On any IO error.
     */
    String getTexture(String material) throws Exception {
        BufferedReader in = new BufferedReader(new FileReader(filename));
        String line = null;
        String pattern = "material " + material;
        while ((line = in.readLine()) != null) {
            // Some OgreXML exports have regex characters in the file name
if (line.matches(pattern.replace("[", "\\[").replace("]", "\\]\")))
{
    while ((line = in.readLine()) != null && !line.startsWith("material")) {
        if (line.matches("\s+texture .*"))
            return (line.trim().split(WHITESPACE)[MATERIAL_INDEX]);
        break;
    }

    // Unable to find an image associated with the material
    return null;
}
package sage.model.loader.ogreXML;

/**
 * {@link OgreXMLVertexBoneAssignment} represents the weight a specific
 * {@link Joint} contributes towards animating a vertex.
 * @author Morgan Darke
 */
public class OgreXMLVertexBoneAssignment implements Comparable<OgreXMLVertexBoneAssignment> {
    private int boneID;
    private float weight;

    /**
     * Constructs a {@link OgreXMLVertexBoneAssignment} with a given
     * {@link Joint} ID and weight.
     * @param ID The ID of the Joint
     * @param weight The weight the joint contributes towards animating a
     * vertex.
     */
    public OgreXMLVertexBoneAssignment(int ID, float weight) {
        this.boneID = ID;
        this.weight = weight;

        if (weight > 1.0f)
            throw new RuntimeException("Error: weight can not be more than 1.0");
    }

    /**
     * Returns the {@link Joint} ID associated with the
     * OgreXMLVertexBoneAssignment.
     * @return The boneID of the OgreXMLVertexBoneAssignment
     */
    public int getBoneID() {
        return boneID;
    }

    /**
     * Returns the weight associated with the OgreXMLVertexBoneAssignment.
     * @return The weight of the OgreXMLVertexBoneAssignment
     */
    public float getWeight() {
        return weight;
    }

    /**
     * This method compares one OgreXMLVertex to another to
     * determine sorting order. Order is based on bone IDs
     */
}
*/
public int compareTo(OgreXMLVertexBoneAssignment o)
{
    if (boneID != o.boneID)
        return boneID - o.boneID;
    else
    {
        if (weight > o.weight)
            return 1;
        else if (weight < o.weight)
            return -1;
        else
            return 0;
    }
}
*/
package sage.scene;

import graphicslib3D.Matrix3D;
import graphicslib3D.Point3D;
import graphicslib3D.Vector3D;

import java.io.File;
import java.io.IOException;
import java.nio.FloatBuffer;
import java.util.ArrayList;
import java.util.TreeMap;

import javax.media.opengl.GL2;
import sage.animation오피스의입니다.
import sage.animation.AnimationKey-frame;
import sage.animation.AnimationSequence;
import sage.animation.Joint;
import sage.renderer.IRenderer;
import sage.renderer.IRenderer.RENDER_METHOD;
import sage.shader.ShaderProgram;
import sage.shader.glsl.GLSLUniform;

import com.jogamp.common.nio.Buffers;
import com.jogamp.opengl.util.texture.Texture;
import com.jogamp.opengl.util.texture.TextureIO;

/**
 * (@link Model3DTriMesh) extends (@link TriMesh) by incorporating support
 * for model animation data and an associated (@link Texture).
 * *
 * @author Morgan Darke
 * *
 */
public class Model3DTriMesh extends TriMesh {

    private static final int MAX_BONES_PER_VERTEX = 4;

    private Texture texture;
    private String textureFileName;
    private ArrayList<Joint> joints;
    private int[] vertexBoneIDs;
    private float[] vertexBoneWeights;
    private FloatBuffer animatedVertices;
    private FloatBuffer animatedNormals;
    private TreeMap<String, AnimationSequence> animations;
    private AnimationSequence currentAnimation;
    private float currentAnimationTime;
    private ShaderProgram shader;
}
private boolean shaderLoaded;

/**
 * Creates a (Model3DTriMesh) with all fields set to null.
 */
public Model3DTriMesh() {
    texture = null;
textureFileName = null;
    joints = null;
    vertexBoneIDs = null;
    vertexBoneWeights = null;
    animatedVertices = null;
    animatedNormals = null;
    animations = null;
currentAnimation = null;
currentAnimationTime = 0;
    shader = null;
    shaderLoaded = false;
}

/**
 * This method gets the collection of AnimationSequence association with
 * the model.
 * @return The collection of animation sequences
 */
public TreeMap<String, AnimationSequence> getAnimations() {
    return animations;
}

/**
 * This method sets the collection of AnimationSequence association with
 * the model.
 * @param animations The collection of animation sequences
 */
public void setAnimations(TreeMap<String, AnimationSequence> animations) {
    this.animations = animations;
}

/**
 * This method gets the buffer of animated normals.
 * @return The buffer of animated normals.
 */
public FloatBuffer getAnimatedNormals() {
    return animatedNormals;
}

/**
 * This method sets the normal buffer that has been transformed by the
 * current animation.
 * @param animatedNormals The buffer of animated normals.
 */
public void setAnimatedNormals(FloatBuffer animatedNormals) {
    this.animatedNormals = animatedNormals;
}

public FloatBuffer getAnimatedVertices() {
    return animatedVertices;
}

public void setAnimatedVertices(FloatBuffer animatedVertices) {
    this.animatedVertices = animatedVertices;
}

public int[] getVertexBoneIDs() {
    return vertexBoneIDs;
}

public void setVertexBoneIDs(int[] vertexBoneIDs) {
    this.vertexBoneIDs = vertexBoneIDs;
}

public float[] getVertexBoneWeights() {
    return vertexBoneWeights;
}

public void setVertexBoneWeights(float[] vertexBoneWeights) {
}
public void setVertexBoneWeights(float[] vertexBoneWeights) {
    this.vertexBoneWeights = vertexBoneWeights;
}

public ArrayList<Joint> getJoints() {
    return joints;
}

public void setJoints(ArrayList<Joint> joints) {
    this.joints = joints;
}

public void setTextureFilename(String filename) {
    textureFileName = filename;
}

public String getTextureFileName() {
    return textureFileName;
}

public Texture getTexture() {
    return texture;
}

*/
public void setTexture(Texture texture) {
    this.texture = texture;
}

/**
 * This method attempts to bind the {@link Texture} associated with this model.
 * @param gl An active GL context
 */
public void bindTexture(GL2 gl) {
    if (texture != null)
    {
        gl.glEnable(GL2.GL_TEXTURE_2D);
        texture.bind(gl);
    }
    else
    {
        gl.glBindTexture(GL2.GL_TEXTURE_2D, 0);
        throw new RuntimeException("Error: attempting to bind null texture.");
    }
}

/**
 * This method unbinds the (@link Texture) associated with this model.
 * @param gl An active GL context
 */
public void unbindTexture(GL2 gl) {
    gl.glBindTexture(GL2.GL_TEXTURE_2D, 0);
}

/**
 * This method loads the (@link Texture) associated with this model.
 * By default the (@link Texture) uses bilinear filtering.
 * @param gl An active GL context
 * @throws IOException If the Texture is unable to be loaded.
 */
public void loadTexture(GL2 gl) throws IOException {
    if (texture == null) {
        texture = TextureIO.newTexture(new File(textureFileName), true);
        gl.glEnable(GL2.GL_TEXTURE_2D);
        gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MIN_FILTER, GL2.GL_LINEAR);
        gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MAG_FILTER, GL2.GL_LINEAR);
    }
gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_WRAP_S, GL2.GL_REPEAT);
gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_WRAP_T, GL2.GL_REPEAT);
}
}

/**
 * This method loads the {@link Texture} associated with this model and gives
 * finer control of how the model's {@link Texture} is filtered.
 *
 * @param gl An active GL context
 * @param minModeFilter The filtering mode when sampling from LOD > 0
 * @param magModeFilter The filtering mode when sampling from LOD 0
 * @param wrapS The address control mode for the S texture coordinate direction
 * @param wrapT The address control mode for the T texture coordinate direction
 * @throws IOException If the Texture is unable to be loaded.
 */
public void loadTexture(GL2 gl, int minModeFilter, int magModeFilter, int wrapS, int wrapT) throws IOException {
    if (texture == null) {
        texture = TextureIO.newTexture(new File(textureFileName), true);
        gl.glEnable(GL2.GL_TEXTURE_2D);
        gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MIN_FILTER, minModeFilter);
        gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MAG_FILTER, magModeFilter);
        gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_WRAP_S, wrapS);
        gl.glTexParameteri(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_WRAP_T, wrapT);
    }
}

/**
 * <code>draw()</code> invokes <code>predraw()</code> for this Leaf,
 * invokes the specified <code>IRenderer</code> to draw the <code>Leaf</code>, then invokes <code>postdraw()</code> for this <code>Leaf</code>.
 */
@Override
public void draw(IRenderer r) {
    this.predraw(r);
    r.draw(this);
    this.postdraw(r);
}

@Override
public boolean predraw(IRenderer r) {
r.setRenderMethod(RENDER_METHOD.VERTEX_ARRAY);
return true;
}

@Override
public void postdraw(IRenderer r) {
    r.setRenderMethod(RENDER_METHOD.IMMEDIATE);
}

/**
 * This method specifies a particular shader to be used to render this model.
 * @param program The shader program.
 */
public void setShaderProgram(ShaderProgram program) {
    shader = program;
}

/**
 * This method returns the ShaderProgram associated with this Model3DTriMesh.
 * @return
 */
public ShaderProgram getShaderProgram() {
    return shader;
}

/**
 * This method sets the attached shader as the current shader program and applies all uniform variables.
 * @param gl An active GL2 context
 */
public void useShader(GL2 gl) {
    if (shader != null) {
        if (!shaderLoaded) {
            shader.init(gl);
            shaderLoaded = true;
        }
        shader.useShader(gl);
        shader.applyUniformVariables(gl);
    }
}

/**
 * This method returns the renderer to the fixed function pipeline.
 * @param gl An Active GL2 context
 */
public void useFixedFunctionPipeline(GL2 gl) {
    if (shader != null)
        shader.useFixedFunction(gl);
}

/**
 * This method returns the current animation time.
 * @return The current animation time.
 */
public float getCurrentAnimationTime() {
    return currentAnimationTime;
}

/**
 * This function gets the uniform variable associated
 * with the specified name.
 * @param name The name of the uniform variable
 * @return The uniform variable
 */
public GLSLUniform<?> getUniform(String name) {
    return shader.getUniform(name);
}

/**
 * This method returns true if the Model3DTriMesh has
 * any animations associated with it.
 * @return true if the Model3DTriMesh has at least one animation sequence
 */
public boolean hasAnimations() {
    return (animations != null && animations.size() > 0);
}

/**
 * This method returns true if the Model3DTriMesh is
 * currently in the middle of an animation sequence.
 * @return true if the Model3dTriMesh is currently being animated.
 */
public boolean isAnimating() {
    return hasAnimations() && currentAnimation != null;
}

/**
 * This method stops any animation that is currently happening and
 * resets the animation time.
 */
public void stopAnimation() {
    currentAnimation = null;
    currentAnimationTime = 0;
}
/**
 * This method changes the animation sequence that is currently being
 * animated and also resets the animation timer.
 * @param name The name of the animation sequence
 */
public void startAnimation(String name) {
    currentAnimation = animations.get(name);
    if (currentAnimation == null)
        throw new RuntimeException("Model does not have any animation with
    the name: " + name);
    currentAnimationTime = 0;
}

/**
 * This method changes the animation sequence that is currently being
 * animated and sets the animation timer to a specific point in time.
 * @param name The name of the animation sequence
 * @param time The time in seconds from the beginning of the animation
 */
public void startAnimation(String name, float time) {
    currentAnimation = animations.get(name);
    if (currentAnimation == null)
        throw new RuntimeException("Model does not have any animation with
    the name: " + name);
    currentAnimationTime = 0;
}

/**
 * This method does the computations for constructing the
 * animRelativeToModelSpace transformations for each Joint.
 */
private void updateAnimationTransformations() {
    // For each Joint, construct the animRelativeToModelSpace transformation
    for (int index = 0; index < joints.size(); index++) {
        Joint joint = joints.get(index);
        ArrayList<AnimationKey-frame> frames = currentAnimation.getKey-frames(index);
        AnimationKey-frame key-frameAnimationMatrix;
        if (frames != null)
            key-frameAnimationMatrix = AnimationKey-frame.find(frames,
currentAnimationTime);
        else
            key-frameAnimationMatrix = new AnimationKey-frame();
Matrix3D toParentSpace = (Matrix3D)
    joint.getFromParentSpace().clone();
    toParentSpace.concatenate(keyframeAnimationMatrix.getTransform());
    joint.setAnimRelativeToParentSpace(toParentSpace);
    if (joint.hasParentJoint()) {
        Matrix3D toModelSpace = (Matrix3D)
            joint.getParent().getAnimRelativeToModelSpace().clone();
        toModelSpace.concatenate(joint.getAnimRelativeToParentSpace());
        joint.setAnimRelativeToModelSpace(toModelSpace);
    } else {
        joint.setAnimRelativeToModelSpace(joint.getAnimRelativeToParentSpace());
    }
    }
}/**
 * This method animates the model by applying a transformation to
 * each vertex and normal. Each vertex can have up to 4 Joint(s)
 * influencing the animation, with each one contributing a specific
 * weight.
 */
private void animate() {
    FloatBuffer vertexBuffer = getVertexBuffer();
    FloatBuffer normalBuffer = getNormalBuffer();
    vertexBuffer.rewind();
    normalBuffer.rewind();
    if (vertexBuffer == null || normalBuffer == null)
        throw new RuntimeException("Error: model has a null vertex and/or
    normal buffer.");
    FloatBuffer transformedVertices =
        Buffers.newDirectFloatBuffer(vertexBuffer.capacity());
    FloatBuffer transformedNormals =
        Buffers.newDirectFloatBuffer(normalBuffer.capacity());
    int[] boneIDs = getVertexBoneIDs();
    float[] weights = getVertexBoneWeights();
    for (int vertex = 0; vertex < vertexBuffer.capacity() / 3; vertex++) {
        Point3D originalVertex = new Point3D(vertexBuffer.get(3 * vertex),
            vertexBuffer.get(3 * vertex + 1), vertexBuffer.get(3 * vertex + 2));
Point3D originalNormal = new Point3D(normalBuffer.get(3 * vertex), normalBuffer.get(3 * vertex + 1), normalBuffer.get(3 * vertex + 2));

// Intermediate weighted vertices and normals
Point3D tempVertex = new Point3D(0, 0, 0);
Point3D tempNormal = new Point3D(0, 0, 0);

// Final weighted vertices and normals
Point3D transformedVertex = new Point3D();
Point3D transformedNormal = new Point3D();

for (int bone = 0; bone < MAX_BONES_PER_VERTEX; bone++) {
    int ID = boneIDs[MAX_BONES_PER_VERTEX * vertex + bone];
    float weight = weights[MAX_BONES_PER_VERTEX * vertex + bone];

    if (weight != 0) {
        Matrix3D matToLocalSpace = joints.get(ID).getFromModelSpace();
        tempVertex = originalVertex.mult(matToLocalSpace.inverse());
        tempNormal = originalNormal.mult(matToLocalSpace.inverse().transpose());

        tempVertex = tempVertex.mult(joints.get(ID).getAnimRelativeToModelSpace());
        tempVertex = tempVertex.mult(weight);

        tempNormal = tempNormal.mult(joints.get(ID).getAnimRelativeToModelSpace());
        tempNormal = tempNormal.mult(weight);

        transformedVertex = transformedVertex.add(tempVertex);
        transformedNormal = transformedNormal.add(tempNormal);
    }
}

transformedVertices.put((float) transformedVertex.getX());
transformedVertices.put((float) transformedVertex.getY());
transformedVertices.put((float) transformedVertex.getZ());

// Normals could need renormalization after multiple transformations
Vector3D renormalizedNormal = new Vector3D(transformedNormal);
renormalizedNormal.normalize();

transformedNormals.put((float) renormalizedNormal.getX());
transformedNormals.put((float) renormalizedNormal.getY());
transformedNormals.put((float) renormalizedNormal.getZ());

} transformedVertices.rewind();
transformedNormals.rewind();

animatedVertices = transformedVertices;
animatedNormals = transformedNormals;
}

/**
 * This method updates the {@link Model3DTriMesh}'s current animation
 * by first updating the animation time and then by selecting the nearest
 * {@link AnimationKey-frame}. Transformations are then applied to update
 * the model's vertices and normals. In the case that the animation time after
 * applying the delta exceeds the length of the animation sequence, this
 * method will loop the animation sequence.
 * @param delta The time in milliseconds to increment the current animation.
 */
public void updateAnimation(float delta) {
    if (isAnimating()) {
        currentAnimationTime += delta / 1000.0f;
        currentAnimationTime %= currentAnimation.getLength();
        updateAnimationTransformations();
        animate();
    }
}
package sage.shader;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import javax.media.opengl.GL2;
import sage.shader.glsl.GLSLUniform;

/**
 * @link ShaderProgram} represents a program that can
 * run on the GPU and is used to do shading. Currently
 * only Vertex Shaders and Pixel/Fragment shaders are
 * supported. Each {@link ShaderProgram} contains an
 * array of Strings for each of the shaders present.
 * 
 * @author Morgan Darke
 * */
public abstract class ShaderProgram {

  protected int vertexShaderID;
  protected int fragmentShaderID;
  protected int shaderProgramID;

  protected String[] vertexSource;

  protected String[] fragmentSource;

  /**
   * This helper function is used to transform the
   * contents of a file into an array of Strings used
   * by the graphics compiler.
   * 
   * @param filename The file to be read
   * @return An array of Strings containing the text in the file
   */
  public String[] loadShaderSource(String filename) {
    StringBuilder builder = new StringBuilder();
    try {
      InputStream in = new FileInputStream(new File(filename));
      BufferedReader reader = new BufferedReader(new InputStreamReader(in));
      String line = null;
      while ((line = reader.readLine()) != null) {
        builder.append(line);
        builder.append('
');
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    return builder.toString().split('
');
  }
}
    in.close();
    }
    catch (Exception e) {
        throw new RuntimeException("Error loading shader source: " +
                filename);
    }

    return new String[] { builder.toString() ];
}

/**
 * This method will initialize the {@link ShaderProgram} by
 * compiling and linking the Vertex and Fragment shaders.
 * Since currently only OpenGL shaders are supported, this
 * method takes a {@link GL2} object as a parameter. This
 * should be changed later to accommodate shaders written
 * in other languages.
 * *
 * @param gl An active GL context
 */
public abstract void init(GL2 gl);

/**
 * This method will make the program the current shader.
 * *
 * @param gl An active GL2 context
 */
public abstract void useShader(GL2 gl);

/**
 * This function will unbind the shader program and the fixed
 * function pipeline will be used instead.
 * *
 * @param gl
 */
public abstract void useFixedFunction(GL2 gl);

/**
 * This function returns the ID used to identify the shader
 * program.
 * *
 * @return the ID of the shader program
 */
public abstract int getProgramID();

/**
 * This function associates a uniform variable with the
 * shader.
 * *
 * @param gl An active GL2 context
 * @param uniform The uniform variable
 */
public abstract void addUniform(GL2 gl, GLSLUniform<?><?> uniform);
/**
 * This function associates a uniform variable with the
 * shader. However, the uniform variable is not yet bound
 * to a location.
 *
 * @param uniform The uniform variable
 */
public abstract void addUniform(GLSLUniform<?> uniform);

/**
 * This function gets the uniform variable associated
 * with the specified name.
 *
 * @param name The name of the uniform variable
 *
 * @return The uniform variable
 */
public abstract GLSLUniform<?> getUniform(String name);

/**
 * This method sets values for all uniform variables used by the shader.
 * In order for the values to be applied, the {@link ShaderProgram} must
 * be the active program.
 *
 * @param gl An active GL context
 */
public abstract void applyUniformVariables(GL2 gl);

/**
 * This method will set the value for the uniform variable with
 * the given name.
 *
 * @param name The name of the variable
 * @param value The value to set
 */
public abstract <E> void setUniformValue(String name, E value);
package sage.shader.glsl;

import java.nio.ByteBuffer;
import java.nio.IntBuffer;
import java.util.HashMap;

import javax.media.opengl.GL2;
import sage.shader.ShaderProgram;

/**
 * @link GLSLShaderProgram] represents a shader written
 * in the GLSL shading language. By default, debugging of
 * shaders is enabled which will print any compiler or
 * linker errors to the console.
 * @author Morgan Darke
 * @param <E>
 */
public class GLSLShaderProgram extends ShaderProgram {

    private static boolean DEBUG_GLSL_SHADERS = true;

    private HashMap<String, GLSLUniform<?>> uniforms;

    public GLSLShaderProgram(String vertexShader, String fragmentShader) {
        uniforms = new HashMap<String, GLSLUniform<?>>();
        vertexSource = loadShaderSource(vertexShader);
        fragmentSource = loadShaderSource(fragmentShader);
    }

    public void init(GL2 gl) {
        attachShaders(gl);
    }

    private void attachShaders(GL2 gl) {
        vertexShaderID = gl.glCreateShader(GL2.GL_VERTEX_SHADER);
        fragmentShaderID = gl.glCreateShader(GL2.GL_FRAGMENT_SHADER);

        // Initialize the status with an error
        IntBuffer status = IntBuffer.allocate(1);
        status.put(0, -1);

        // Compile Vertex Shader
        gl.glShaderSource(vertexShaderID, 1, vertexSource, null, 0);
        gl.glCompileShader(vertexShaderID);
        gl.glGetShaderiv(vertexShaderID, GL2.GL_COMPILE_STATUS, status);

        if (!status.get(0) == GL2.GL_TRUE) {
            if (DEBUG_GLSL_SHADERS)
                getCompilerError(gl, vertexShaderID);

            throw new RuntimeException("Compile error in vertex shader.");
        }
    }
}
// Compile Fragment Shader
gl.glShaderSource(fragmentShaderID, 1, fragmentSource, null, 0);
gl.glCompileShader(fragmentShaderID);
gl.glGetShaderiv(fragmentShaderID, GL2.GL_COMPILE_STATUS, status);

if (status.get(0) != GL2.GL_TRUE) {
    if (DEBUG_GLSL_SHADERS)
        getCompilerError(gl, fragmentShaderID);

    throw new RuntimeException("Compile error in fragment shader.");
}

shaderProgramID = gl.glCreateProgram();

// Link Shader program
gl.glAttachShader(shaderProgramID, vertexShaderID);
gl.glAttachShader(shaderProgramID, fragmentShaderID);
gl.glLinkProgram(shaderProgramID);

gl.glValidateProgram(shaderProgramID);

if (DEBUG_GLSL_SHADERS)
    getLinkError(gl);
}

/**
 * This method will make the program the current shader.
 *
 * @param gl An active GL2 context
 */
public void useShader(GL2 gl) {
    gl.glUseProgram(shaderProgramID);
}

/**
 * This function will unbind the shader program and the fixed
 * function pipeline will be used instead.
 *
 * @param gl
 */
public void useFixedFunction(GL2 gl) {
    gl.glUseProgram(0);
}

/**
 * This function returns the ID used to identify the shader
 * program.
 *
 * @return the ID of the shader program
 */
public int getProgramID() {
    return shaderProgramID;
}

/**
 * This function associates a uniform variable with the
* shader.
* 
* @param gl An active GL2 context
* @param uniform The uniform variable
*/
public void addUniform(GL2 gl, GLSLUniform<? extends GLSLUniform> uniform) {
    uniform.bindToShader(gl, this);
    uniforms.put(uniform.getName(), uniform);
}

/**
 * This function associates a uniform variable with the
 * shader. However, the uniform variable is not yet bound
 * to a location.
 * 
 * @param uniform The uniform variable
 */
public void addUniform(GLSLUniform<? extends GLSLUniform> uniform) {
    uniform.bindToShader(this);
    uniforms.put(uniform.getName(), uniform);
}

/**
 * This function gets the uniform variable associated
 * with the specified name.
 * 
 * @param name The name of the uniform variable
 * @return The uniform variable
 */
public GLSLUniform<? extends GLSLUniform> getUniform(String name) {
    GLSLUniform<? extends GLSLUniform> uniform = uniforms.get(name);
    if (uniform == null)
        throw new RuntimeException("No uniform variable with that name: " + name);
    return uniform;
}

/**
 * This method will toggle debugging information for
 * the shader on or off. If debugging is enabled the
 * compile or link error will be printed to the console.
 * 
 * @param debug true if debugging information is to be enabled,
 * false otherwise.
 */
public static void enableDebugging(boolean debug) {
    DEBUG_GLSL_SHADERS = debug;
}

/**
 * This method will check if the shader compiled
 * correctly and will print out any errors to
 * the console.
 * 
 * @param gl An active GL2 context
 * @param shaderID the ID of the vertex or fragment shader
 */
private void getCompilerError(GL2 gl, int shaderID) {
    IntBuffer intBuffer = IntBuffer.allocate(1);
    gl.glGetShaderiv(shaderID, GL2.GL_COMPILE_STATUS, intBuffer);
    if (intBuffer.get(0) != 1) {
        gl.glGetShaderiv(shaderID, GL2.GL_INFO_LOG_LENGTH, intBuffer);
        int size = intBuffer.get(0);
        System.err.println("Program compile error: ");
        if (size > 0) {
            ByteBuffer byteBuffer = ByteBuffer.allocate(size);
            gl.glGetShaderInfoLog(shaderID, size, intBuffer, byteBuffer);
            for (byte b : byteBuffer.array())
                System.err.print((char) b);
            System.err.flush();
        } else {
            System.err.println("Unknown compile error");
            System.err.flush();
        }
    }
    System.exit(1);
}

/**
 * This method will check if the shader linked correctly and will print out any errors to
 * the console.
 * @param gl An active GL2 context
 */
private void getLinkError(GL2 gl) {
    IntBuffer intBuffer = IntBuffer.allocate(1);
    gl.glGetProgramiv(shaderProgramID, GL2.GL_LINK_STATUS, intBuffer);
    if (intBuffer.get(0) != 1) {
        gl.glGetProgramiv(shaderProgramID, GL2.GL_INFO_LOG_LENGTH, intBuffer);
        int size = intBuffer.get(0);
        System.err.println("Program link error: ");
        if (size > 0) {
            ByteBuffer byteBuffer = ByteBuffer.allocate(size);
            gl.glGetProgramInfoLog(shaderProgramID, size, intBuffer, byteBuffer);
            for (byte b : byteBuffer.array())
                System.err.print((char) b);
            System.err.flush();
        } else {
            System.err.println("Unknown link error");
            System.err.flush();
        }
    }
}
System.err.print((char) b);
}
else
    System.out.println("Unknown link error");

    System.exit(1);
}

/**
 * @param name
 * @param value
 */

public <E> void setUniformValue(String name, E value) {
    @SuppressWarnings("unchecked")
    GLSLUniform<E> uniform = (GLSLUniform<E>) getUniform(name);
    uniform.setValue(value);
}
}
package sage.shader.glsl;

import javax.media.opengl.GL2;
import sage.shader.ShaderProgram;

/**
 * GLSLUniform describes a GLSL uniform variable which has a name, type, and value and is associated with a particular ShaderProgram.
 * @author Morgan Darke
 * @param <E> The type of data stored by the uniforms, which can be integers, floats, matrices, or arrays of any of the previous types.
 */
public abstract class GLSLUniform<E> {
    protected String name;
    protected int location;
    protected GLSLShaderProgram shader;
    protected E value;

    /**
     * Constructs a new GLSLUniform with the given name.
     * @param name The name of the uniform variable
     */
    public GLSLUniform(String name) {
        this.name = name;
        location = -1;
    }

    /**
     * This method associates a uniform variable with a particular ShaderProgram. The ShaderProgram must also be the active program. However, the location of the variable nor its value are set.
     * @param shader The shader program
     */
    public void bindToShader(GLSLShaderProgram shader) {
        this.shader = shader;
    }

    /**
     * This method associates a uniform variable with a particular ShaderProgram. The ShaderProgram must also be the active program.
     */
}
public void bindToShader(GL2 gl, GLSLShaderProgram shader)
{
    this.shader = shader;
    this.location = gl.glGetUniformLocation(shader.getProgramID(), name);
}

/**
 * This method associates the uniform with a particular {link GLSLShaderProgram},
 * and sets an initial value. The shader program must also be the
 * active program.
 * @param gl An active GL2 context
 * @param shader The shader program
 * @param value The initial value
 */
public void bindToShader(GL2 gl, GLSLShaderProgram shader, E value)
{
    this.shader = shader;
    this.location = gl.glGetUniformLocation(shader.getProgramID(), name);
    this.setValue(gl, value);
}

/**
 * This method gets the name of the uniform variable.
 * @return The name of the uniform variable
 */
public String getName()
{
    return name;
}

/**
 * This method gets the location of the uniform variable
 * @return The location of the uniform variable
 */
public int getLocation()
{
    return location;
}

/**
 * This method sets the uniform variable with the given value
 * immediately. In order for the values to be applied, the
 * {link ShaderProgram} must be the active program.
 * @param gl An active GL2 context
 * @param value The value to set
 */
public abstract void setValue(GL2 gl, E value);

/**
* This method sets the new value of the uniform variable
* which will be applied on the next render pass. Uniforms
* of array types need to override this method to do array size
* checks.
* @param value The value to set
*/
public void setValue(E value) {
    this.value = value;
}

/**
* This method sets the uniform variable with the current value.
* @param gl An active GL2 context
*/
public void apply(GL2 gl) {
    // The value could have been set outside a GL context, so make sure
    // location is valid.
    if (location == -1)
        location = gl.glGetUniformLocation(shader.getProgramID(), name);
    setValue(gl, this.value);
}

/**
* This method converts an array of Floats into
* an array of primitive types.
* @param array The array of Float objects.
* @return an array of float primitives.
*/
protected float[] toFloatPrimitiveArray(Float[] array) {
    float[] primitive = new float[array.length];
    for (int i = 0; i < primitive.length; i++)
        primitive[i] = array[i];
    return primitive;
}

/**
* This method converts an array of Integers into
* an array of primitive types.
* @param array The array of Integer objects.
* @return an array of int primitives.
*/
protected int[] toIntegerPrimitiveArray(Integer[] array) {
    int[] primitive = new int[array.length];
    for (int i = 0; i < primitive.length; i++)
        primitive[i] = array[i];
    return primitive;
}
package sage.shader.glsl;

import javax.media.opengl.GL2;

/**
 * {@link GLSLUniform1f} represents a uniform variable
 * with a single float value.
 * @author Morgan Darke
 */
public class GLSLUniform1f extends GLSLUniform<Float>{

    /**
     * Constructs a {@link GLSLUniform1f} with the given name.
     * @param name The name of the uniform.
     */
    public GLSLUniform1f(String name) {
            super(name);
    }

    public void setValue(GL2 gl, Float value) {
            gl.glUniform1f(location, value);
    }
}
package sage.shader.glsl;

import javax.media.opengl.GL2;

/**
 * {@link GLSLUniform1i} represents a uniform variable with a single integer value. {@link GLSLUniform1i}
 * can also be used to set values for Sampler uniform variables.
 *
 * @author Morgan Darke
 */

public class GLSLUniform1i extends GLSLUniform<Integer> {

    /**
     * Constructs a {@link GLSLUniform1i} with the given name.
     *
     * @param name The name of the uniform.
     */
    public GLSLUniform1i(String name) {
        super(name);
    }

    public void setValue(GL2 gl, Integer value) {
        gl.glUniform1i(location, value);
    }
}
package sage.shader.glsl;

import javax.media.opengl.GL2;
import graphicslib3D.Matrix3D;

/**
 * {@link GLSLUniformMatrix4f} represents a uniform variable
 * of one 3x3 matrices of float values.
 * @author Morgan Darke
 */
public class GLSLUniformMatrix4f extends GLSLUniform<Matrix3D> {

    /**
     * Constructs a {@link GLSLUniformMatrix4f} with the given name.
     * @param name The name of the uniform.
     */
    public GLSLUniformMatrix4f(String name) {
        super(name);
    }

    @Override
    public void setValue(GL2 gl, Matrix3D value) {
        gl.glUniformMatrix4fv(location, 1, false, value.getFloatValues(), 0);
    }
}
APPENDIX B

OgreXML Document Type Definitions

<!ELEMENT mesh (sharedgeometry?, sub-meshes, skeletonlink?, boneassignments?, levelofdetail?, submeshnames?, poses?, animations?, extremes?)>
<!ELEMENT sharedgeometry (vertexbuffer+)>
<!ATTLIST sharedgeometry vertexcount CDATA #REQUIRED>
<!ELEMENT sub-meshes (submesh+)>
<!ELEMENT submesh (textures?, faces, geometry?, boneassignments?)>
<!ATTLIST submesh material CDATA #REQUIRED>
usesharedvertices (true|false) "true" use32bitindexes (true|false) "false" operationtype (triangle_list|triangle_strip|triangle_fan) "triangle_list">
<!ELEMENT textures (texture+)>
<!ELEMENT texture EMPTY>
<!ATTLIST texture alias CDATA #REQUIRED name CDATA #REQUIRED>
<!ELEMENT faces (face+)>
<!ATTLIST faces count CDATA #IMPLIED>
<!ELEMENT face EMPTY>
<!ATTLIST face v1 CDATA #REQUIRED v2 CDATA #IMPLIED v3 CDATA #IMPLIED>
<!ELEMENT geometry (vertexbuffer+)>
<!ATTLIST geometry vertexcount CDATA #IMPLIED >
<!ELEMENT skeletonlink EMPTY>
<!ATTLIST skeletonlink name CDATA #REQUIRED>
<!ELEMENT boneassignments (vertexboneassignment+)>
<!ELEMENT boneassignment EMPTY>
<!ATTLIST vertexboneassignment vertexindex CDATA #REQUIRED boneindex CDATA #REQUIRED weight CDATA "1.0">
<!ELEMENT levelofdetail ( (lodmanual|lodgenerated)+ )>
<!ATTLIST levelofdetail strategy CDATA "Distance" numlevels CDATA #REQUIRED manual (true|false) "false">
<!ELEMENT lodmanual EMPTY>
<!ATTLIST lodmanual value CDATA #REQUIRED>
<![ELEMENT lodgenerated (lodfacelist)>  
<!ATTLIST lodgenerated  value CDATA #REQUIRED  
meshname CDATA #REQUIRED  >

<!ELEMENT lodfacelist (face)+>  
<!ATTLIST lodfacelist  submeshindex CDATA #REQUIRED  
numfaces CDATA #REQUIRED  >

<!ELEMENT vertexbuffer (vertex+)>  
<!ATTLIST vertexbuffer  positions (true|false) "false"  
normals (true|false) "false"  
colours_diffuse (true|false) "false"  
colours_specular (true|false) "false"  
texture_coords  
texture_coord_dimensions_0 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_1 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_2 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_3 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_4 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_5 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_6 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
texture_coord_dimensions_7 = 
  (1|2|3|4|float1|float2|float3|float4|short1|short2|short3|short4|ubyte4|colour|colour_argb|colour_abgr) "2"  
tangents (true|false) "false"  
tangent_dimensions (3|4) "3"  
binormals (true|false) "false"  
</ELEMENT vertex (position, normal?, tangent?, binormal?, colour_diffuse?, colour_specular?,  
texcoord*)>

<!ELEMENT position EMPTY>  
<!ATTLIST position  x CDATA #REQUIRED  
y CDATA #REQUIRED  
z CDATA #REQUIRED  >

<!ELEMENT normal EMPTY>  
<!ATTLIST normal  x CDATA #REQUIRED  
y CDATA #REQUIRED  
z CDATA #REQUIRED  >

<!ELEMENT tangent EMPTY>  
<!ATTLIST tangent  x CDATA #REQUIRED  


<!ELEMENT binormal EMPTY>
<!ATTLIST binormal
    x CDATA #REQUIRED
    y CDATA #REQUIRED
    z CDATA #REQUIRED>
<!ELEMENT colour_diffuse EMPTY>
<!-- 'value' is a space-separated string containing r,g,b and optionally alpha
    for example value="1.0 0.0 0.0 0.5" or value="0.7 0.5 0.2" -->
<!ATTLIST colour_diffuse
    value CDATA #REQUIRED>
<!ELEMENT colour_specular EMPTY>
<!-- 'value' is a space-separated string containing r,g,b and optionally alpha
    for example value="1.0 0.0 0.0 0.5" or value="0.7 0.5 0.2" -->
<!ATTLIST colour_specular
    value CDATA #REQUIRED>
<!ELEMENT texcoord EMPTY>
<!ATTLIST texcoord
    u CDATA #REQUIRED
    v CDATA "0"
    w CDATA "0"
    x CDATA "0">
<!ELEMENT submeshnames (submeshname+)> 
<!ELEMENT submeshname EMPTY>
<!ATTLIST submeshname
    name CDATA #REQUIRED
    index CDATA #REQUIRED>
<!ELEMENT poses (pose+)>
<!-- A single pose references a single set of geometry data with a set of
    offsets.
    If target is 'mesh', targets the shared geometry, if target is submesh,
    targets
    the submesh identified by 'index'.
    -->
<!ELEMENT pose (poseoffset+)>
<!ATTLIST pose
    target (mesh|submesh) #REQUIRED
    index CDATA "0"
    name CDATA "">
<!-- poseoffset lists the vertices that change position, and by how much -->
<!ELEMENT poseoffset EMPTY>
<!ATTLIST poseoffset
    index CDATA #REQUIRED
    x CDATA #REQUIRED
    y CDATA #REQUIRED
    z CDATA #REQUIRED>
<!ELEMENT animations (animation+)>
<!ELEMENT animation (tracks)>
<!ATTLIST animation
    name CDATA #REQUIRED
    length CDATA #REQUIRED>
<!ELEMENT tracks (track+)>
<!ELEMENT track (key-frames)>
<!-- Morph animation is a key-framed set of absolute vertex positions. Cannot
    be blended with other morph animations or pose animation.
    Pose animation is a set of key-frames referencing poses and a weight,
    with one track per set of vertex data.
    Can be blended with other poses but not with morph animation. -->
If target is 'mesh', targets the shared geometry, if target is submesh, targets
the submesh identified by 'index'.

```xml
<!ATTLIST track
  target (mesh|submesh) #REQUIRED
  index CDATA "0"
  type (morph|pose) #REQUIRED>
```

Key-frames are applicable for all tracks, but for morph tracks they contain positions, and for pose tracks they contain pose references.

```xml
<!ELEMENT key-frames (key-frame*)>
<!ELEMENT key-frame (position*, poseref*)>
<!ATTLIST key-frame
time CDATA #REQUIRED>
```

Pose reference, links to pose via numeric index. Target of parent track must agree with target of referenced pose. For a single track, each key-frame can reference multiple poses at different weights.

```xml
<!ELEMENT poseref EMPTY>
<!ATTLIST poseref
  poseindex CDATA #REQUIRED
  influence CDATA "1.0">
```

Optional extremity points on sub-meshes for sub-object transparency sorting.

```xml
<!ELEMENT extremes (submesh+>)
<!ELEMENT submesh_extremes (position+>)
<!ATTLIST submesh_extremes
  index CDATA #REQUIRED>
```
<!ELEMENT skeleton (bones, bonehierarchy, animations?, animationlinks?) >
<!ELEMENT bones (bone+) >
<!ELEMENT bone (position, rotation, scale?) >
<!ATTLIST bone
    id CDATA #REQUIRED
    name CDATA #REQUIRED>
<!ELEMENT position EMPTY>
<!ATTLIST position
    x CDATA #REQUIRED
    y CDATA #REQUIRED
    z CDATA #REQUIRED>
<!ELEMENT rotation (axis)>
<!ATTLIST rotation
    angle CDATA #REQUIRED>
<!ELEMENT axis EMPTY >
<!ATTLIST axis
    x CDATA #REQUIRED
    y CDATA #REQUIRED
    z CDATA #REQUIRED>
<!ELEMENT bonehierarchy (boneparent*)>
<!ELEMENT boneparent EMPTY>
<!-- NB both the below are bone names, not ids -->
<!ATTLIST boneparent
    bone CDATA #REQUIRED
    parent CDATA #REQUIRED>
<!ELEMENT animations (animation+)>
<!ELEMENT animation (tracks)>
<!ATTLIST animation
    name CDATA #REQUIRED
    length CDATA #REQUIRED>
<!ELEMENT tracks (track+)>
<!ELEMENT track (key-frames)>
<!ATTLIST track
    bone CDATA #REQUIRED>
<!ELEMENT key-frames (key-frame+)>
<!ELEMENT key-frame
    time CDATA #REQUIRED>
<!ELEMENT translate EMPTY>
<!ATTLIST translate
    x CDATA #REQUIRED
    y CDATA #REQUIRED
    z CDATA #REQUIRED>
<!ELEMENT rotate (axis)>
<!ATTLIST rotate
    angle CDATA #REQUIRED>
<!ELEMENT scale EMPTY>
<!-- UNIFORM SCALE is 'factor', or you can use per-axis (not both) -->
<!ATTLIST scale
    factor CDATA #IMPLIED
    x CDATA #IMPLIED
    y CDATA #IMPLIED
    z CDATA #IMPLIED>
<!ELEMENT animationlinks (animationlink+)>
<!ELEMENT animationlink EMPTY>
<!ATTLIST animationlink
    skeletonName CDATA #REQUIRED
    scale CDATA "1.0">
REFERENCES


