sc-dance Library Doumentation (Draft)

IOANNIS ZANNOS

July 21, 2025

Contents

1 Introduction

1.1 Purpose and Features

The ‘sc-dance‘ library is a SuperCollider framework designed for networked live coding of dance and animation, primarily interfacing with the Godot game engine. It provides tools to:

- Load and manage motion capture (MOCAP) session data.
- Control 3D avatars and their animations in Godot via OSC (Open Sound Control).
- Process and filter animation data in real-time.
- Generate sound from animation data, enabling expressive audiovisual performances.
- Offer a modular and extensible architecture for custom animation and sound synthesis.

2 Getting Started

2.1 Installation in SuperCollider

To install ‘sc-dance‘ in SuperCollider, clone the repository into your SuperCollider Extensions directory. You can find your Extensions directory by evaluating ‘Platform.userAppSupportDir ++ “Extensions”‘ in SuperCollider.

```
cd /path/to/SuperCollider/Extensions
git clone https://github.com/your-repo/sc-dance.git
```

After cloning, recompile the SuperCollider class library by selecting ‘Language -> Recompile Class Library‘ in the SuperCollider IDE.

2.2 2Loading and Starting the Demo Project in Godot

The ‘sc-dance‘ library is designed to work with a companion Godot project. A recommended demo project is $\text{boy}_{\text{andbirds}}$.

1. Clone the ‘$\text{boy}_{\text{andbirds}}$‘ repository into a suitable location on your computer.

```
git clone https://github.com/iani/boy_and_birds.git
```

2. Open the Godot Engine. From the project manager, select 'Import' and navigate to the cloned 'boy$_{\text{and birds}}$' directory. Select the 'project.godot' file.

3. Once imported, open the 'boy$_{\text{and birds}}$' project in Godot.

4. Run the Godot project. This will start the 3D environment and prepare it to receive OSC messages from SuperCollider.

5. In SuperCollider, evaluate the initial setup code (e.g., 'Avatar.load.valuesGui.sendToGodot.play;' from the example files) to establish the connection and begin sending animation data.

3 Architecture of the sc-dance SuperCollider Library

The 'sc-dance' library is structured to provide a clear separation of concerns, managing animation data, OSC communication, and sound synthesis. Key components include:

3.1 Class Overview

- **Adapter/**: Classes related to adapting data formats and handling changes.
  - 'Adapter.sc': Base class for data adaptation.
  - 'AdapterController.sc': Manages multiple adapters.
  - 'ObjectAdapterAPI.sc': Provides an API for object adaptation.
  - 'TraceChange.sc': Utility for tracing changes in adapted data.
- **Animation/**: Core classes for animation data handling.
  - 'AnimationController.sc': Manages the flow of animation data, polling control buses and publishing values. It orchestrates the interaction between 'Avatar' and 'Animator'.
  - 'Animator.sc': Responsible for filtering and publishing animation messages, applying transformations and filters to raw MOCAP data.

  - ‘Avatar.sc‘: Represents a single animated figure. It manages session data, animators, and controllers, and handles OSC communication for a specific avatar. (See detailed description below).
  - ‘AvatarSettings.sc‘: Configuration settings for an avatar.
  - ‘Joint.sc‘: Represents a single joint of an avatar, providing methods for accessing and processing its motion data (e.g., ‘slopePhrase‘).
  - ‘JointFilter.sc‘: Applies filters to individual joint data.
  - ‘JointIO.sc‘: Manages input/output for joint data.
  - ‘OscFile.sc‘: Handles reading/writing OSC data to/from files.
  - ‘OscRecorder.sc‘: Records OSC messages.
  - ‘OscRecorderPath.sc‘: Manages paths for OSC recordings.
  - ‘OscSequence.sc‘: Manages sequences of OSC messages.
  - ‘OscSessions.sc‘: Manages OSC sessions.
  - ‘Props.sc‘: Handles properties of an avatar (e.g., color, type, position, rotation).
  - ‘RokokoParser.sc‘: Parses Rokoko MOCAP data.
  - ‘SessionData.sc‘: Manages loading and accessing recorded MOCAP session data.
- **CodeDisplay/**: Utility for displaying code.
  - ‘ShowCode.sc‘: Displays SuperCollider code.
- **ControlAndSound/**: Classes related to control signals and sound generation.
  - ‘JointSlope.sc‘: Calculates the slope of joint movements.
  - ‘plusSymbolJointShortcuts.sc‘: Adds shortcuts for joint-related operations to the ‘Symbol‘ class.
  - ‘SynthFuncTemplate.sc‘: Base class for creating reusable synth functions.
  - ‘PlayBufTemplate.sc‘: (New) Subclass of ‘SynthFuncTemplate‘ for playing buffers.
- **GUI/**: Classes for graphical user interfaces.
  - ‘guiOnCloseMethods.sc‘: Methods for GUI window closing events.

  - ‘plusWindowBounds.sc‘: Adds methods for managing window bounds.
  - ‘Windows.sc‘: Utility for creating and managing GUI windows.

- **ObjectInstanceManagement/**: Manages named instances of objects.
  - ‘NamedInstance.sc‘: Base class for objects that can be named and managed globally.

- **OSC/**: Classes for OSC communication.
  - ‘OscControl.sc‘: Manages global OSC reception.
  - ‘plusArraySendLocal.sc‘: Adds methods for sending OSC messages locally.
  - ‘plusForwardMsg.sc‘: Adds methods for forwarding OSC messages.
  - ‘plusMainRecvOscFunc.sc‘: Adds methods for handling main OSC receive functions.
  - ‘plusObjectAsOscMessage.sc‘: Adds methods for converting objects to OSC messages.
  - ‘TraceOsc.sc‘: Utility for tracing OSC messages.

- **PathsAndDataLoading/**: Manages file paths and data loading.
  - ‘PathBookmark.sc‘: Base class for managing bookmarked paths.
  - ‘plusStringPathNameMethods.sc‘: Adds methods for string path manipulation.
  - ‘AvatarAssets.sc‘: (Renamed from ‘RokokoSessionsBookmark‘) Manages session data and assets.

- **PatternShortcuts/**: Provides shortcuts for SuperCollider Patterns.
  - ‘PatternShortcuts.sc‘: Collection of pattern-related shortcuts.

3.2 The Avatar Class: How it all hangs together

The ‘Avatar‘ class is central to the ‘sc-dance‘ library, acting as the primary interface for controlling and interacting with a single animated figure. It brings together various components to manage the avatar’s state, animation, and sound generation.

3.2.1 Inheritance.

Avatar inherits from NamedInstance, allowing multiple avatars to be created and referenced by unique names (e.g., `Avatar('myDancer')`).

3.2.2 Key Instance Variables:

- ‘sessionData‘: An instance of ‘SessionData‘ that holds the loaded motion capture data for the avatar.
- ‘animator‘: An instance of ‘Animator‘ responsible for processing and filtering the raw animation data before it’s sent to Godot or used for sound.
- ‘controller‘: An instance of ‘AnimationController‘ that manages the control buses for the avatar’s joints and handles the polling of these buses to update the ‘animator‘.

3.2.3 Core Functionality:

- **Loading Data**: ‘Avatar‘ can load MOCAP session data from files, which is then managed by its ‘sessionData‘ instance.
- **OSC Communication**: It handles enabling/disabling remote control from MOCAP software (like Rokoko Studio) and forwarding animation data to Godot via OSC.
- **Animation Playback**: It orchestrates the playback of loaded animation data through its ‘animator‘ and ‘controller‘.
- **Sound Synthesis Integration**: Through its ‘controller‘, ‘Avatar‘ allows you to attach SuperCollider synths to individual joint data. The ‘addSynth‘ method (and the newly added ‘setSynthCtl‘) enables dynamic control of these synths based on animation parameters.
- **Joint Access**: It provides convenient access to individual ‘Joint‘ objects, allowing direct manipulation or querying of joint-specific data (e.g., ‘~hip.slopePhrase‘).

3.2.4 Relationship with other classes:

- ‘Avatar‘ instantiates and manages ‘SessionData‘, ‘Animator‘, and ‘AnimationController‘.

- ‘AnimationController‘ in turn creates and manages ‘Joint‘ and ‘JointIO‘ instances for each joint of the avatar.
- ‘Animator‘ uses ‘JointFilter‘ instances to process joint data.
- ‘AvatarAssets‘ (formerly ‘RokokoSessionsBookmark‘) is used by ‘Avatar‘ to manage session paths and asset loading.

In essence, ‘Avatar‘ acts as the central hub for a single character, coordinating the flow of animation data from source (file or live MOCAP) through processing and filtering, to its final output as visual animation in Godot and sound in SuperCollider.

4 Example Files

The ‘sc-dance‘ library includes a ‘Guides‘ folder (and its subfolders) containing various example ‘.scd‘ files. These examples demonstrate different aspects of the library’s functionality, from basic setup to advanced sound synthesis and animation control.

This section outlines the topics covered by these examples and suggests a logical progression for exploring the library.

4.1 Loading and Playing Animations:

- How to load pre-recorded MOCAP session data.
- Basic playback of animations in Godot.
- Controlling animation speed and looping.

4.2 Modifying and Synthesizing Animations:

- Applying filters to animation data (e.g., smoothing, scaling).
- Real-time manipulation of joint data.
- Creating custom animation behaviors.

4.3 Making Sound from Animation Data:

- Connecting animation parameters to sound synthesis (e.g., joint position to pitch, joint velocity to amplitude).

- Using ‘slopePhrase‘ to extract meaningful control signals from motion.
- Granular synthesis driven by animation data.
- Triggering events based on animation thresholds.

4.4 Creating Avatars with Different Names and the Role of the Default Avatar:

- Instantiating multiple ‘Avatar‘ objects.
- Managing and switching between active avatars.
- Understanding the concept of the “default” avatar and its implications for global control.

4.5 Advanced Topics (Suggested Future Examples):

- Live MOCAP integration (if applicable).
- Custom UGen development for animation processing.
- Interfacing with other external software.
- Building complex multi-avatar scenes.