AnimationMixer

The AnimationMixer is a core class in Three.js for managing animations within a 3D scene. It handles the timing, playback, and combination of animation sequences. An AnimationMixer can manage multiple animations simultaneously and allows, for example, starting, stopping, or crossfading between movements.

In the context of the workshop pipeline, the AnimationMixer becomes relevant when external animation clips (e.g., glTF or FBX animations) are to be used. In the workshop, movements are primarily implemented procedurally using JavaScript code and are therefore not yet implemented using the AnimationMixer.

Typical usage pattern:

```
const mixer = new THREE.AnimationMixer(model);
const action = mixer.clipAction(animationClip);
action.play();
```

Animation clips

Animation clips are stored motion sequences of a 3D model. They contain time-defined changes to properties such as position, rotation, scaling, or blendshape values. An animation clip can contain, for example:

- Walking movements
- Jumping animations
- Dance sequences
- Facial expressions
- Hand gestures

In Three.js, animation clips are typically played back together with an AnimationMixer.

In this workshop, animations will first be implemented directly in JavaScript as bone rotations. This has the architectural advantage that the motion description is separated from the application logic, making it easier to understand and adapt. For more complex movements, such as natural walking or dancing animations, pre-prepared animation clips are generally used in professional pipelines.

Avatar

An avatar is the digital representation of a user, character, or artificial entity within a virtual environment. Avatars can assume various roles, such as personal identity, game character, virtual assistant, or digital performer.

In this workshop, the term specifically refers to a humanoid 3D model created with VRoid Studio, exported as a VRM file, and then displayed in a browser.

An avatar typically comprises several layers:

- geometric model
- textures and materials
- skeletal structure (rig)
- facial animations
- metadata
- interaction logic

In more advanced concepts, such as virtual idol or metaverse systems, an avatar can be further enhanced with dialogue systems, AI components, voice, and autonomous behavior mechanisms.

Avaturn

Avaturn is a tool for creating realistic 3D avatars. The platform allows the generation of personalized characters, often based on photos or parametric adjustments. Unlike VRoid Studio, which focuses heavily on stylized anime and manga characters, Avaturn pursues a semi-realistic to realistic character style.

In the workshop, Avaturn is mentioned as a possible alternative to VRoid Studio. Since Avaturn can export avatars as VRM-compatible models, among other things, a similar browser pipeline can, in principle, be built using Three.js and @pixiv/three-vrm.

Typical differences from VRoid Studio:

VRoid Studio	**Avaturn**
Anime style	realistic style
parametric character design	photo/ template-oriented generation
direct VRM orientation	multiple export options

Blendshapes

Blendshapes are a method for deforming 3D geometry. Alternative states of a model are defined and interpolated. A blendshape can, for example, represent:

- open mouth
- smile
- closed eyes
- raised eyebrows

During runtime, these target states can be blended with different weightings.

In VRM avatars, blendshapes play a central role in:

- facial expressions
- lip sync
- viseme animations
- emotional representation

This becomes particularly relevant in the workshop when the avatar speaks using text-to-speech. Blendshape values are used to simulate mouth movements.

Bone-Rotation

A bone-Rotation describes the rotation of a single bone within a rig system. Humanoid 3D models consist internally of a hierarchical bone structure. Each bone has a position and orientation relative to its parent node. Typical bones include:

- Head
- Chest
- Left Upper Arm
- Right Lower Leg
- Hips

In the workshop, animations are created directly using bone rotations:

```
rightUpperArm.rotation.z += 0.5;
```

his allows gestures, movements, and postures to be generated programmatically.

Bone rotations form the technical foundation of many animation techniques:

- Procedural Animation
- Keyframe Animation
- Motion Capture Retargeting
- Inverse Kinematics

CORS (Cross-Origin Resource Sharing)

CORS is a security mechanism in modern web browsers that regulates access to resources from different origins. An origin is essentially determined by:

- Protocol
- Hostname
- Port

By default, browsers prevent many connections between different origins. In this workshop, CORS becomes relevant when:

- VRM files are loaded,
- external textures are embedded,
- local files are opened directly via `file://`.

For this reason, a local web server is used in the workshop.

Typical error scenario:

```
Access to fetch blocked by CORS policy
```

Using a local server, e.g. via VS Code Live Server, Node.js or Python HTTP Server, avoids this problem..

CSS (Cascading Style Sheets)

CSS is the standard technology for designing websites. While HTML defines the structure of an application, CSS handles its visual presentation. Typical CSS tasks include:

- Colors
- Layout
- Spacing
- Sizes
- Responsive Design
- Animations
- User Interface Design

In this workshop, CSS will be used to:

- Design the control panel,
- Format buttons,
- Define dialog boxes,
- Arrange overlay elements on the 3D canvas.

```
button {
    border-radius: 10px;
    background: white;
}
```

Therefore, CSS forms a highly relevant component of the web-based runtime environment alongside HTML, JavaScript, and Three.js.

Facial Expressions

Facial Expressions refer to the representation of a digital avatar's facial expressions. They serve to visually depict emotions, reactions, or communicative nuances. Typical facial expressions include:

- Smiling
- Sadness
- Surprise
- Anger
- Blinking
- Thoughtfulness

Technically, facial expressions are implemented in many avatar systems using blendshapes. This involves activating pre-defined facial shapes with varying degrees of emphasis.

VRM avatars often already have predefined facial expressions.

In this workshop, the topic will be particularly relevant for:

- Talking avatars
- Lip sync
- Emotional dialogue systems
- Virtual idol productions

Simple control can be achieved, for example, via:

```
expressionManager.setValue("happy", 1.0);
```

glTF / glTF 2.0

glTF (GL Transmission Format) is an open file format for the efficient storage and transmission of 3D content. Developed by the Khronos Group, it is now considered one of the most important exchange formats for real-time 3D applications. A glTF model can contain, among other things:

- Geometry
- Materials
- Textures
- Animations
- Cameras
- Light sources
- Scene structure

glTF 2.0 is the currently most widely used major version.

glTF is particularly important in the workshop because VRM is technically based on glTF 2.0.

VRM can therefore be understood simply as:

glTF 2.0
+ humanoides Rig
+ Blendshapes
+ Avatar-meta data
+ VRM-spezific extensions

Three.js loads glTF files typically via the `GLTFLoader`.

HTML (HyperText Markup Language)

HTML is the standard markup language of the World Wide Web. HTML defines the structure of a web application. Typical HTML elements are:

- Headings
- Forms
- Buttons
- Input fields
- Containers

- Dialogs
- Canvas elements

In this workshop, HTML forms the basis of the browser application. It defines, for example:

- Upload buttons for VRM files
- Toolbars
- Text input for voice and ELIZA
- Dialog window for voice selection
- The canvas for displaying the 3D scene

Example:

```
<canvas id="scene"></canvas>
```

The canvas element serves as a drawing surface for Three.js and thus for displaying the avatar.

HTML therefore handles the structural organization of the entire user interface.

Humanoid-Rig

A humanoid-rig is a standardized skeletal structure for humanoid 3D figures. It defines which bones a model possesses and how they are hierarchically organized. Typical elements of a humanoid rig are:

- Hips
- Spine
- Chest
- Head
- Upper Arm
- Lower Arm
- Hand
- Upper Leg
- Lower Leg
- Foot

Using a standardized rig facilitates:

- Animations
- Reusability
- Motion capture integration
- Exchange between platforms

VRM defines its own humanoid rig schema. This allows code to access the desired bone regardless of the platform:

```
vrm.humanoid.getNormalizedBoneNode("rightUpperArm")
```

In the workshop, the humanoid rig is the basis for all movement animations.

JavaScript

JavaScript is the core programming language of modern web applications. It runs directly in the browser and handles the dynamic control of user interfaces, data processing, and interactions.

In this workshop, JavaScript acts as the execution logic layer for the entire pipeline. Among other things, it controls:

- Three.js initialization
- Loading VRM files
- Avatar animations
- Camera logic and avatar transformations
- Text-to-speech
- ELIZA dialog system
- Background and scene control

A typical example:

```
button.addEventListener("click", () => {
    speak(textInput.value);
});
```

JavaScript thus forms the link between:

HTML →	Structure
CSS →	Design
Three.js	3D rendering
VRM	avatar design

Without JavaScript, the pipeline would simply be a static website.

LookAt behavior

The LookAt behavior describes an avatar's ability to dynamically direct its gaze toward a target object. This typically involves adjusting head, eye, or neck movements:

- Avatar looks at the camera
- Avatar follows the mouse pointer
- Avatar looks at a conversation partner
- Avatar focuses attention on objects

VRM supports LookAt mechanisms by default. The @pixiv/three-vrm library provides functions for this. A simplified concept is, for example :

Determine target point
→ Calculate gaze direction
→ Adjust head/eye rotation

LookAt behavior contributes significantly to the social credibility of virtual characters.

In the workshop, this can be expanded upon, for example, through:

- Gaze toward the speaking person
- Gaze behavior in the dialogue system
- Camera-guided attention
- • * Virtual idol performance scenarios

Node.js

Node.js is a JavaScript runtime environment outside of the web browser. While JavaScript was originally primarily executed within browsers, Node.js allows the same language to be used for:

- Server applications
- Development tools
- Build processes
- Local development environments

In this workshop, Node.js will be primarily relevant as a tool within the development environment. Typical use cases include:

- Local web server
- Package management via npm
- Project initialization
- Development tools

A simple local server, for example, can be deployed using Node-based tools. Node.js is therefore not part of the avatar itself, but rather supports the pipeline's technical infrastructure.

npm (Node Package Manager)

npm is the standard package manager for Node.js. It is used to install, update, and manage external software libraries. In modern web projects, npm is used to conveniently integrate dependencies. Examples include:

- Three.js
- Vite
- Express
- TypeScript
- React

A typical npm command is: (here, the installation of the Three package)

```
npm install three
```

In this workshop, the application will initially be built using a CDN to keep the learning curve low. However, in more advanced projects, npm can enable more professional project management.

npx

npx is a tool for directly executing Node packages. Unlike npm, an application does not need to be permanently installed globally.

Example (here: starting the Vite tool without requiring prior global installation):

```
npx vite
```

npx is frequently used for:

- Project generators
- Development servers
- Build tools
- One-off utilities

In more advanced versions of the Workshop pipeline, npx can be helpful when setting up a more modern development environment.

Phoneme

Phoneme are the smallest meaning-distinguishing sounds in a language. Examples in German:

- a
- e
- m
- p
- sch

Speech is created from phonemes in succession.

In avatar systems, phonemes are relevant for:

- Speech synthesis
- Lip sync
- Mouth animation

Since 3D models cannot represent acoustic sounds, phonemes are often translated into visemes.

Phoneme
→ speech sound
Viseme
→ visual mouth shape

This becomes relevant in the workshop when text-to-speech is combined with talking avatars.

Polygon modelling

Polygon modelling is a fundamental technique in 3D computer graphics. It involves constructing objects from polygonal surfaces. The most common polygon shapes are:

- Triangles
- Quadrilaterals

A 3D avatar ultimately consists of many interconnected polygons. These define:

- Body shape
- Face
- Clothing
- Hair
- Accessories

In tools like VRoid Studio, polygon modeling is largely parametric and user-friendly. The user primarily works via sliders and visual adjustments rather than manual mesh editing. Polygon modeling forms the geometric basis of every VRM avatar.

Rule-Based Chatbot

A rule-based chatbot generates dialogues based on predefined rules and pattern recognition. Unlike modern generative AI systems, it does not use large language models. A typical process is:

Analyze input
→ Recognize pattern
→ Select appropriate rule
→ Generate response

Example:

User: "I am sad."
*Rule: "I am **"*
Antwort: "Why are you sad?"

This concept will be demonstrated in the workshop using ELIZA.

Render loop

The render loop is the continuous update loop of a real-time 3D application. In each iteration, the following steps are typically performed:

Update time
→ Calculate animations
→ Update scene state
→ Render image
→ Request next frame

In Three.js, this is usually implemented with:

```
requestAnimationFrame()
```

Example:

```
function animate() {
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
```

In the workshop, the render loop handles, among other things:

- Avatar updates
- Animations
- LookAt behavior
- VRM updates
- Scene rendering

The render loop thus forms the temporal core of the application.

Rigging

Rigging refers to the process of equipping a 3D model with a controllable skeletal structure. Bones are defined and connected to the model's geometry. A rig then allows for:

- Movements
- Animations
- Body postures
- Facial control

The typical process is:

```
3D model
→ Create skeleton
→ Connect bones
→ Define weights
→ Animatable character
```

In VRoid Studio, rigging is generated automatically. This means that exported VRM avatars already have a functional humanoid rig. Therefore, no complete manual rigging needs to be performed in the workshop to still utilize its results for animations and bone rotations.

TTS (Text-to-Speech)

Text-to-Speech (TTS) refers to methods for automatically converting written text into synthetic speech. A TTS system typically processes:

Text input
→ linguistic analysis
→ sound generation
→ audio signal

TTS systems can have varying levels of quality, from simple synthetic voices to nearly natural-sounding AI-generated voices.

In this workshop, TTS will be used to make the avatar speak. The text output will come from sources such as:

- free text input
- ELIZA dialogues
- potentially AI dialogue systems in the future

The actual speech synthesis takes place in the browser pipeline via the WebSpeech API.

Three.js

Three.js is a widely used JavaScript library for developing 3D web applications. It significantly simplifies access to WebGL and abstracts many of the technical details of modern 3D graphics programming. Three.js provides, among other things:

- Scene management
- Camera models
- Lighting
- Materials
- Geometries
- Animation
- Asset loader

A typical Three.js structure consists of:

Scene
→ Camera
→ Renderer
→ Render-Loop

Example:

```
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera();
const renderer = new THREE.WebGLRenderer();
```

In this workshop, Three.js takes on the role of the central rendering engine. It controls:

- 3D scene
- Avatar display
- Lighting
- Camera

- Textures
- Floor and wall surfaces
- Real-time updates

Three.js/ three-vrm

The combination Three.js/ three-vrm refers to the collaboration between the general 3D rendering library Three.js and the VRM-specific extension three-vrm. While Three.js can process generic 3D objects, three-vrm adds functionality for humanoid avatar models. This combination enables:

Loading the VRM
→ Interpreting the avatar
→ Using the humanoid rig
→ Controlling expressions
→ Executing look-at behavior

This combination forms the technical core of the selected workshop pipeline.

@pixiv/three-vrm

@pixiv/three-vrm s a specialized library for using VRM avatars within Three.js. Originally developed by Pixiv, it extends Three.js with VRM-specific features such as:

- VRM import
- Humanoid access
- Expressions
- LookAt system
- Metadata processing
- Avatar updates

Typical import:

```
import { VRM } from "@pixiv/three-vrm";
```

n this workshop, @pixiv/three-vrm is a central core library. It provides the technical bridge between::

VRM file ↔ Three.js scene

Without this library, a significant portion of the VRM-specific logic would have to be implemented manually.

Visemes

A viseme describes the visual representation of a speech sound. While a phoneme represents an acoustic sound, a viseme describes its visible mouth shape, for example:

→ A wide open mouth
→ O rounded lips
→ M closed lips

Avatar systems use visemes to generate lip synchronization (lip sync). Typical implementation::

Text
→ Phonemes
→ Visemes
→ Blendshape control

This concept will become relevant in the workshop when speech synthesis and facial animation are more closely integrated.

VRM

VRM is a standardized file format for humanoid 3D avatars. Technically, VRM is based on glTF 2.0, but extends it with avatar-related properties. VRM adds, among other things, the following to glTF:

- Humanoid Rig
- Blendshapes
- Expressions
- LookAt configuration
- Metadata
- Usage information

VRM = glTF 2.0 + Avatar extensions

In the workshop, VRM serves as the central exchange format between:

VRoid Studio
→ Browser application
→ Three.js / three-vrm

The standardization of VRM facilitates interoperability between different tools and platforms.

VRoid Studio

VRoid Studio is a tool for creating humanoid 3D avatars. Its focus is on user-friendly, visually guided character design. This allows users to create avatars without in-depth 3D modeling knowledge. VRoid Studio allows, among other things:

- Facial design
- Body customization
- Hairstyles
- Clothing
- Colors
- Material parameters

The avatars can then be exported directly as VRM files.

In the workshop, VRoid Studio serves as the starting point of the pipeline.

VS Code (Visual Studio Code)

VS Code is a widely used development environment for software and web development. It supports numerous programming languages and extensions.

In this workshop, VS Code will be recommended for:

- HTML editing
- CSS editing
- JavaScript development
- Project organization
- Local testing

Helpful extensions include:

- Live Server
- Syntax highlighting
- JavaScript tools
- Git integration

Web Server

A web server provides files and applications via HTTP or HTTPS. Browser-based applications are typically not loaded directly from the file system (`file://`) but are delivered via a server.

In this workshop, a local web server is important for:

- VRM file loading
- Module imports
- CORS avoidance
- Web API compatibility

Possible options::

- VS Code Live Server
- Node.js-Server
- Python HTTP Server

The browser then typically accesses this local web server by selecting a specific port number:

```
http://localhost:8000
```

WebSpeech API

The WebSpeech API is a browser-based interface for speech processing. It comprises two main areas:

- Speech Synthesis
- Speech Recognition

The workshop pipeline primarily uses the Speech Synthesis API. It enables:

- Voice selection
- Speech synthesis
- Speech settings
- Speech rate
- • Pitch control

```
const utterance = new SpeechSynthesisUtterance("Hallo Welt");
speechSynthesis.speak(utterance);
```

In the workshop, the WebSpeech API forms the basis for:

- Speaking avatars
- ELIZA response output
- Experimental speech interfaces

In the future, Speech Recognition could also be integrated to enable speech input instead of text input.