engine

Table of contents

Supported platforms

- Windows
- Mac OS X
- Linux
  - Extra dependencies: libz, libpng, SDL 1.2 and SDL_Mixer 1.2, libtheora, libogg, libvorbis
  - Development dependencies (Ubuntu): libsdl1.2-dev libsdl-mixer1.2-dev libtheora-dev npm openjdk-8-jdk doxygen git
- MorphOS
  - Extra dependencies: PowerSDL

Recommended setup

- It's really up to the demo coder but roughly following is needed from engine's perspective:
  - 1.5 GHz+ processor with 2 or more cores
  - GPU with shader support
  - 1 GB RAM

Portability and restrictions

- Use only ASCII characters in string handling, file and directory names. Avoid using whitespaces.
- Use only OGG audio files with sampling rate 11025, 22050 or 44100 Hz
- OGV (Theora) videos should preferably be as fixed framerate. Variable framerates could go out-of-sync
  - Only first 30 seconds of single video will be loaded
  - You can use ffmpeg to make clips, examples:
    - Copy ogg video: ffmpeg -ss 00:00:10 -i 0807_Camera_Thrills_of_the_War_07_00_59_00_3mb.ogv -t 00:00:15 -c copy camera_thrills_clip.ogv
    - Cut ogg video and reencode without audio: ffmpeg -ss 00:02:29 -i Dividean1943.ogv -t 00:00:03 -vb 1M -an planning.ogv
- Avoid using more than 2 GB of RAM simultaneously
  - Some operating systems might not allow too much RAM per process
  - Videos are currently buffered to RAM so large videos can easily reach 2GB limit
- Script and data/ directory must have case sensitive file name (i.e. don't have file name "Paska.png" in script when in data directory it's actually "PASKA.PNG")
- Engine is mainly using OpenGL fixed pipeline (a.k.a. "legacy OpenGL")
- Engine supports 32bit, 64bit, little and bid endian architectures. Ensure that code will support it as well.

MorphOS portability

- MorphOS machines are in general significantly slower than normal machines where demos will run so take that into account :)
- Shaders, VBOs and FBOs are not supported
  - [MorphOS 3D]

Command line switches

- --muteSound - Force mute engine audio
- --changePosition <time string, f.e., "1:30.5"> - Define player's start position
- --resolution <width>x<height> - Force resolution
- --fullscreen <1 or 0> - Force fullscreen ON(1)/OFF(0)
- --noMenu - Don't display start menu
- --verbose - Verbose log output (log available in Windows stdout.txt and stderr.txt; other platforms STDOUT/STDERR)
- --tool - Tool mode for demo making (implicitly enables --verbose)
- --demoPath &ltpath> - Set the root of the demo data directory

Demo editor tool mode

- CTRL+HOME - Move demo timer to the begin (0:00)
- CTRL+END - Move demo timer to the end and pause demo

- CTRL+1 - Move demo timer backward one second
- CTRL+2 - Move demo timer forward one second
- CTRL+3 - Pause or unpause demo
- CTRL+5 - Shallow refresh demo - Code, shader and script files are refreshed but data resources (images, 3d objects) are not refreshed
- SHIFT+CTRL+5 - Deep refresh demo - Code, shader, script files and data resources (images, 3d objects) are refreshed
- CTRL+p - Take a screenshot
- Tabulator - Show latest screen log.
  - Screen log will contain ERROR type log entries.
  - You can add print custom debug text with screenPrint("text here") function call.
- Saving shader (.vs/.fs) or javascript file (.js) in text editor will force the specific file to be refreshed on-the-fly. If this is attempted in too short time span then the refresh might not always happen. In those cases use shallow/deep refresh instead.

GNU Rocket integration

In tool mode engine will automatically attempt to connect to GNU Rocket. If GNU Rocket server connection can't be established then player mode will be automatically used. Track files must be located in following place and naming convention: data/sync/sync_<TRACK NAME>.track

Exporting to video

You can use .kkapture in Windows for exporting demos to video.

Fonts

Custom font

- Custom font file path: data/font.png
- Font file can be any width and height
- Font characters must be monospaced so that there are 10 characters per row and 10 rows with even width and height
- Font's characters will be read in ASCII order
  - First character is space ' ' (ASCII dec 32)
  - Last character is tilde '~' (ASCII dec 126)

Default font

- Default font is the same as the font used in the selection menu
- Default font's character has pixel dimensions 9x13
- If no custom font is provided then default font is used

Demo scripting

- Demo scripting is based on JSON and JavaScript/ECMAScript 5
- Demo scripting consists of two major parts
  - Demo definitions - Demo static definitions such as resource files and setup information
  - Scripting language - Actual script files

Javascript extensions

- Matrix.js
- Vector.js

Demo definitions

data/js/script.js contains the main demo definition data.

- music <filename> - Location to the main music file (OGG Vorbis)
- beatsPerMinute <bpm> - The average beats per minute - default value 120
- rowsPerBeat <rpb> - Rows per beat in GNU Rocket - default value 1

- totalTime <time string> - Demo total playing time
- useInput - If true then engine will send system and input device (keyboard, mouse...) events to javascript. Default is false.
- kanttuCompatibility <true/false> - If engine should try to emulate cross compatibility with Kanttu's demo engine (a.k.a. Jumalauta compatibility mode). Default is false.
    - This has an effect in 2d coordinate system - images (position x:0,y:0) are centered to the bottom left corner origo).
    - This has an effect to .3ds file handling.
        - Clipping plane handling
        - Camera handling
        - More to come as they are discovered...
- window - Window settings
    - title <title> - Window title
- screen - Screen settings - 2D dimensions default to 1280x720
    - width <width> - The 2D maximum width of the screen
    - height <height> - The 2D maximum height of the screen
    - aspectRatio - The aspect ratio settings for screen and 3D, defaults to 16:9 aspect ratio
        - width <width> - Width aspect
        - height <height> - Height aspect
- clearColor - Sets the main screen clear color
    - r <double> - red - default value 0.0
    - g <double> - green - default value 0.0
    - b <double> - blue - default value 0.0
    - a <double> - alpha - default value 0.0
- shaders - List of all shaders
    - Individual shader
        - name <name> - Name of the shader
        - filename <filename> - File location of the shader
- shaderPrograms - List of all shader programs
    - Individual shader program
        - name <name> - Name of the shader program
        - shaders - List of shaders attached to the shader program
            - Individual shader
                - name <name> - Name of the shader
- effects - List of demo source files
    - Individual source file
        - name <effect name> - Reference name of the source file
        - filename <filename> - File location of the source file
    - If ommitted engine will create effect "Demo" that references to file data/js/Demo.js
- scenes - List of scenes in the demo
    - Individual scene
        - name <name> - Name of the scene
        - effect <effect name> - The source file / effect reference
        - startTime <time string> - Start time of the scene, if empty then defaults to "0:00"
        - durationTime <time string> - End time of the scene, if empty then defaults to totalTime
    - If ommitted engine will create scene "Demo" with effect "Demo"

Example script.js:

```
Settings.demoScript = {
  "music": "data/champ.ogg",
  "beatsPerMinute": 90,
  "rowsPerBeat": 1,
  "totalTime": "2:10",
  "kanttuCompatibility": false,
  "useInput": true,
  "window":
  {
    "title": "Jumalauta!"
  },
  "screen":
  {
     "width": 640
    ,"height": 480
    ,"aspectRatio":
    {
       "width": 16
      ,"height": 10
    }
  },
  "clearColor": {
    "r": 0.0,
    "g": 0.0,
    "b": 0.0,
    "a": 0.0
  },
  "shaders": [
    {
      "name": "Deformation",
      "filename": "data/shader/deformation.fs"
    }
  ],
  "shaderPrograms": [
    {
      "name": "Deformation",
      "shaders": [
        {
          "name": "Deformation"
        }
      ]
    }
  ],
  "effects": [
    {
      "name": "Jazz",
      "reference": "data/js/Jazz.js"
    }
  ],
  "scenes": [
    {
      "name": "Jazz",
      "effect": "Jazz",
      "startTime": "0:00",
      "durationTime": "2:10"
    }
  ]
};
```

NOTE: In script.js it is possible to override default loading bar by defining function: Loader.drawLoadingBar = function(percent) { /*OpenGL functions to draw loaderbar goes here here*/ }

Scripting language

- Here are the examples and documenation of the JSON based scripting language that can be used in the JavaScript files.

Scripting language reference

- Here's documented scripting language reference.
- Scripting language should primarily be defined in scene class' init() method.
- Script is rendered with method call: player.drawAnimation(this.loader.animationLayers);

addSync

- addSync defines a sync pattern that can be applied in the addAnimation definition
  - Sync point times redefine the time of specific animation primitive where sync is applied to.
  - When GNU Rocket is used for syncing it defines the sync timing and values

```
//inhouse sync pattern
Sync.addSync([
{
     "name":"pattern1"              //name of the sync pattern that is referenced in the addAnimati
    ,"start":"0:00"                 //relative start time of whole sync pattern
    ,"duration":"0:08"              //duration time of the whole sync pattern
    ,"end":"0:08"                   //relative end time of whole sync pattern
    ,"syncStartFunction":<function> //sync start function - initialized on every start of sync poin
    ,"syncEndFunction":<function>   //sync end function - initialized on every end of sync point
    ,"syncRunFunction":<function>   //sync run function - initialized on every time sync point is a
    ,"pattern": [      //sync pattern sync point times
         {
             /*Sync pattern point 1*/
            ,"start":"0:00"                 //relative start time of the sync point
            ,"duration":"0:01"              //duration time of the sync point
            ,"end":"0:00"                   //relative start time of the sync point
            ,"syncStartFunction":<function> //sync start function - initialized on every start of s
            ,"syncRunFunction":<function>   //sync run function - initialized on every time sync po
        }
        ,{/*Sync pattern point 2*/}
        ,{/*Sync pattern point ...N*/}
    ]
}]);

//GNU Rocket sync track
Sync.addSync([
{
     "name":"track.variable"        //name of the GNU Rocket sync track that is referenced in the a
    ,"type":"rocket"                //Defines that sync pattern handled by GNU Rocket
    ,"syncStartFunction":<function> //sync start function - initialized once
    ,"syncRunFunction":<function>   //sync run function - initialized on every time sync point is a
}]);
```

addAnimation

- addAnimation method adds animation definition

```
this.loader.addAnimation([
     {/*Animation JSON 1*/}
    ,{/*Animation JSON 2*/}
    ,{/*Animation JSON ...N*/}
]);
```

- To process all added animation you need to call Loader.processAnimation() (preferably in scene's init() or postInit())
- In case you have not defined postInit() function then Loader.processAnimation() will be automatically called after init() function.

```
this.loader.processAnimation();
```

- To calculate and draw all processed animations you need to call Player.drawAnimation(Loader.animationLayers) (in scene's run())
- In case you have not defined run() function then Player.drawAnimation() will be automatically called.

```
this.player.drawAnimation(this.loader.animationLayers);
```

Animation type static definitions

- Main animation types and non generic object related definitions
- Image & video (2D/3D)

```
 "image": <PNG/OGV file path>
 {
     "canvasWidth":<width>             //the 2D screen relative width
    ,"canvasHeight":<height>           //the 2D screen relative height
    ,"perspective":<perspective>       //defines if the image should be rendered in 2D or 3D mode, d
  ,"align":<alignment>               //Sets the image alignment:
                                       //1 - center alignment
                                       //2 - horizontal / X-axis alignment
                                       //3 - vertical / Y-axis alignment
  ,"uv": {                           //Define UV coordinates of the image
     "uMin": 0.0
    ,"vMin": 0.0
    ,"uMax": 1.0
    ,"vMax": 1.0
  }
 }
```

- Video parameters can be defined in image tag

```
 "image": {
    "name": "video.ogv",
    "video": {
       "loop": <loop>    //1 = LOOP, 0 = do not loop. Default is 0.
      ,"fps": <fps>      //overrides videos normal fps. can be used for skipping frames etc...
      ,"speed": <speed>  //Can adjust the speed of playback. 1.0 is default.
    }
 }
```

- Text (2D/3D)

```
"text":
{
    "string":<string>            //text that should be rendered
   ,"perspective":<perspective>  //defines if text should be rendered in 2D or 3D mode, default "2
}
,"clearDepthBuffer":<boolean>    //default false
,"align":<alignment>             //Sets the image alignment:
                                   //1 - center alignment
                                   //2 - horizontal / X-axis alignment
                                   //3 - vertical / Y-axis alignment
```

- Object (3D)

```
 "object":<3DS file path or object name>  //name of the 3D object or path to .3ds file
,"shape":                                 //basic shape - for rendering default shapes
 {
   "type":<basic shape type>              //MATRIX (not object but for global transformation animat
                                          //CUBE
                                          //CYLINDER
                                          //DISK
                                          //SPHERE
 }
,"objectFunction":<function>              //custom JavaScript object drawing function
,"clearDepthBuffer":<boolean>             //default false
,"fps":<decimal>                          //animation frames per second
,"frame":<decimal>                        //animation display constant frame
```

- FBO (2D/3D)

```
"fbo":
{
    "name":<fbo name> //name of the fbo - NOTE: fbo can be used in "image" animation by referring
   ,"action":<action>
       //actions:
       //"begin" - start writing to FBO
       //"unbind" - end writing to FBO and return to the main screen rendering
       //"draw" - draw the FBO
       //"end" - perform "unbind" and "draw" operations
   ,"width":<width> //FBO texture width
   ,"height":<height> //FBO texture height
   ,"storeDepth":<true/false> //If FBO should store depth values as well. Default is false.
   "dimension":[ //adjust the FBO render quality by changing the render dimensions
       {
            "x":1.0 //FBO render width in percentage 0.0 - 1.0 - default is 1.0
           ,"y":1.0 //FBO render height in percentage 0.0 - 1.0 - default is 1.0
       }
   ]
}
```

- Light (3D)

```
"light":
{
    "index":0         //light index number
   ,"action":<action> //action "begin" - light ON; "end" - light OFF
}
,"diffuseColor":[ //light's ambient color
    {
        "r":255  //color red   - accepts values 0-255 - default for light 0 is 255, others 0
       ,"g":255  //color green - accepts values 0-255 - default for light 0 is 255, others 0
       ,"b":255  //color blue  - accepts values 0-255 - default for light 0 is 255, others 0
       ,"a":255  //color alpha - accepts values 0-255 - default is 255
    }
    ,{/*color animation primitive...N*/}
]
```

```
,"ambientColor":[ //light's ambient color
    {
        "r":255  //color red   - accepts values 0-255 - default is 0
       ,"g":255  //color green - accepts values 0-255 - default is 0
       ,"b":255  //color blue  - accepts values 0-255 - default is 0
       ,"a":255  //color alpha - accepts values 0-255 - default is 255
    }
    ,{/*color animation primitive...N*/}
]
,"specularColor":[ //light's ambient color
    {
        "r":255  //color red   - accepts values 0-255 - default for light 0 is 255, others 0
       ,"g":255  //color green - accepts values 0-255 - default for light 0 is 255, others 0
       ,"b":255  //color blue  - accepts values 0-255 - default for light 0 is 255, others 0
       ,"a":255  //color alpha - accepts values 0-255 - default is 255
    }
    ,{/*color animation primitive...N*/}
]
```

- Camera (3D)
  - There can be only one active camera in the main scene/FBO

```
"camera": <camera name>
//where camera is looking at
,"target":[
    {
        "x":0.0 //camera look-at X, default 0
       ,"y":0.0 //camera look-at Y, default 0
       ,"z":0.0 //camera look-at Z, default 0
    }
    ,{/*target animation primitive...N*/}
]
//camera's up vector
,"up":[
    {
        "x":0.0 //camera up vector X, default 0
       ,"y":1.0 //camera up vector Y, default 1
       ,"z":0.0 //camera up vector Z, default 0
    }
    ,{/*target animation primitive...N*/}
]
//camera's perspective setup
,"perspective":[
   {
       "fov":45       //view Y angle in degrees, default 45
       ,"aspect":16/9 //aspect ratio, default 16:9
       ,"near":1      //near clipping plane
       ,"far":1000    //far clipping plane
   }
   ,{/*perspective animation primitive...N*/}
]
,"cameraRelativePosition": <object name> //lock camera's position to track 3d object main type
,"cameraRelativeTarget": <object name>   //lock camera's target(look-at) to track 3d object main t
```

- Time definitions

```
"start":<time>              //start time, default is the scene's start time
"duration":<time>           //duration time, default is the scene's duration time
"end":<time>                //end time, default is the scene's end time
```

- Layer - defines the rendering order in alphabetical order

```
"layer":<layer>      //layer, default value 1, layers can be in range 1 - 99999
```

- Custom functions

```
,"initFunction":<function>  //custom JavaScript function that is called during addAnimation method 
,"runFunction":<function>   //custom JavaScript function that is called every time animation is ren
```

- Shader definitions

```
"shader":
{
    "name":<shader name>, //shader name as defined in script.js. alternatively you can directly ref
    "variable":[
         {
             "name":<name>   //uniform variable name as defined in the shader code
            ,"type":<type>   //uniform variable type (int or float) as defined in the shader code,
            ,"value":[<value1>,<...valueN>} //1-4 values of the uniform as an array
        }
        ,{/*variable...N*/}
    ]
}
```

- Sync - apply defined sync pattern to the animation
    - Note: This notation is not mandatory for GNU Rocket tracks. If GNU Rocket track is used then stick with 0.0 - 1.0 value range as engine interprets the track strictly as progress percent.
- By default sync is applied to whole pattern but all or some can be excluded

```
"sync":{
     "name":<patter name> //string name of the pattern that was defined in addSync method
    ,"all":<boolean>      //apply sync default value to all animation calculations - default value
    //single enable/disable animation types. Default values are based on "all" variable.
    ,"color":<boolean>
    ,"angle":<boolean>
    ,"position":<boolean>
    ,"scale":<boolean>
}
```

Animation primitives

- First animation primitive array element omits time definitions and uses defaults always
- All animation primitives support time definitions and arrays
- After first animation primitive, the primitives always inherit the values from previous primitive
- Non-time related animation variables support JavaScript dynamic injection

```
"animationPrimitive": [
    { //animation primitive 1
        "start":<time>     //start time, default is the animation block's start time
        ,"duration":<time> //duration time, default is the animation block's duration time
        ,"end":<time>      //end time, default is the animation block's end time
        /*more animation variables per animation primitive*/
    }
     ,{/*animation primitive 2*/}
     ,{/*animation primitive 3*/}
     /*...*/
     ,{/*animation primitive N*/}
]

"scale": [
    {
         "x":1.0          //scale X value 1.0 = 100% - default is 1.0
        ,"y":1.0          //scale Y value 1.0 = 100% - default is 1.0
    ,"z":1.0          //scale Z value 1.0 = 100% - default is 1.0
    ,"uniform2d":1.0 //scale X & Y = uniform2d value - default undefined
    ,"uniform3d":1.0 //scale X, Y & Z = uniform2d value - default undefined
    }
]

//in case of images:
// - if position is not given then image is aligned to center
// - 2d image's origo (x:0,y:0) is bottom-left corner so that image is centered to origo.
"position": [
    {
         "x":0.0  //position X - default is context specific but usually 0.0
        ,"y":0.0  //position Y - default is context specific but usually 0.0
        ,"z":0.0  //position Z - default is context specific but usually 0.0
    }
]

//image and object pivot point
"pivot": [
  {
     "x":0.0  //position X - default is 0.0
    ,"y":0.0  //position Y - default is 0.0
    ,"z":0.0  //position Z - default is 0.0
  }
```

```
]

"color": [
    {
         "r":255  //color red   - accepts values 0-255 - default is 255
        ,"g":255  //color green - accepts values 0-255 - default is 255
        ,"b":255  //color blue  - accepts values 0-255 - default is 255
        ,"a":255  //color alpha - accepts values 0-255 - default is 255
    }
]

"angle": [
    {
         "degreesX":0  //3d angle degrees X - default is 0
        ,"degreesY":0  //3d angle degrees Y - default is 0
        ,"degreesZ":0  //2d/3d angle degrees Z - default is 0
        ,"x":1.0       //3d rotation vector X coordinate - default is 1.0
        ,"y":1.0       //3d rotation vector Y coordinate - default is 1.0
        ,"z":1.0       //3d rotation vector Z coordinate - default is 1.0
]
```

JavaScript dynamic injection

- Instead of normal JSON definition "variableName":1.0 or "variableName":getConstantValue() it's possible to inject dynamic JavaScript code with "{}" notation.
- Example: "variableName":"{return javaScriptVariable*Math.sin(getSceneTimeFromStart()/10.0);}"

Scripting language examples

2D image animation examples

```
//Move jml_fist.png from origo (x:0,y:0) to other end of the screen in 10 seconds
this.loader.addAnimation([{
     "start": "0:00", "duration": "5:00"
    ,"layer": 1, "image": "data/jml_fist.png"
    ,"position": [
          {"x":0, "y":0}
         ,{"duration":"0:10", "x":getScreenWidth(), "y":getScreenHeight()}
    ]
}]);

//Play jml_fist.ogv animation and move from origo (x:0,y:0) to other end of the screen in 10 second
this.loader.addAnimation([{
   "start": "0:00", "duration": "5:00"
  ,"layer": 1, "image": "data/jml_fist.ogv"
  ,"position": [
      {"x":0, "y":0}
     ,{"duration":"0:10", "x":getScreenWidth(), "y":getScreenHeight()}
  ]
}]);

//Wait 2 seconds and then:
//1) vertically "flip" jml_fist.png by scaling y from 1.0 to -1.0
//2) horizontally "flip" the image
//3) "flip" image back to normal
this.loader.addAnimation([{
     "start": "0:00", "duration": "5:00"
    ,"layer": 1, "image": "data/jml_fist.png"
    ,"scale": [
          {"x":1, "y":1}
         ,{"start":2, "duration":1, "x":1, "y":-1}
         ,{"duration":1, "x":-1, "y":-1}
         ,{"duration":1, "x":1, "y":1}
    ]
}]);

//Rotate jml_fist.png clock-wise 360 degrees in 3 seconds
this.loader.addAnimation([{
     "start": "0:00", "duration": "5:00"
    ,"layer": 1, "image": "data/jml_fist.png"
    ,"angle": [
         {"degreesZ":0}
        ,{"duration":"0:03", "degreesZ":360}
    ]
}]);

//Rotate jml_fist.png counter clock-wise 720 degrees pivoting from the center of screen in 10 secon
this.loader.addAnimation([{
     "start": "0:00", "duration": "5:00"
    ,"layer": 1, "image": "data/jml_fist.png"
    ,"position": [{"x":getScreenWidth()/2, "y":0}]
  ,"pivot":[{"y":getScreenHeight()/2}]
    ,"angle": [
         {"degreesZ":0}
        ,{"duration":"0:10", "degreesZ":-360*2}
    ]
}]);

//1) Make jml_fist.png completely black and transparent
//2) Transition the image from completely transparent to completely opaque in 2.5 seconds
//3) Transition the image from completely black to white in 5 seconds
this.loader.addAnimation([{
     "start": "0:00", "duration": "5:00"
    ,"layer": 1, "image": "data/jml_fist.png"
    ,"color": [
         {"r":0, "g":0, "b":0, "a":0}
        ,{"duration":"0:02.5", "a":255}
        ,{"duration":"0:05", "r":255, "g":255, "b":255}
    ]
}]);

//Example where jml_fist.png is rotated and scaled down in a spiral-like manner. After the spiral i
this.loader.addAnimation([{
     "start": "0:00", "duration": "5:00"
    ,"layer": 1, "image": "data/jml_fist.png"
    ,"position": [
         {"x":getScreenWidth()/2, "y":0}
        ,{"duration":"0:25","y":getScreenHeight()/2}
    ]
  ,"pivot": [
     {"y":getScreenHeight()/2}
    ,{"duration":"0:25", "y":0}
  ]
    ,"angle": [
         {} //use default values == 0.0
        ,{"duration":"0:25", "degreesZ":-360*5}
        ,{"duration":"0:50", "degreesZ":-360*25}
    ],
    "color": [
         {}
        ,{"start":"0:25", "duration":"0:30", "a":0}
    ],
    "scale": [
         {"x":2.5,"y":2.5}
        ,{"duration":"0:25", "x":1,"y":1}
        ,{"duration":"0:50", "x":20,"y":20}
    ]
}]);
```

Text drawing examples

Draw 2D text to the screen:

```
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30"
    ,"layer": 100, "text":{"string":"Here is a 2D text string with\na line break!"}
    ,"scale": [
          {"x":0.5,"y":0.5}
    ]
    ,"color": [
          {"g":0,"b":0}
    ]
    ,"angle": [
           {"degreesZ":0}
          ,{"duration":"0:3", "degreesZ":360}
    ]
}]);
```

Draw 3D text to the screen:

```
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30"
    ,"layer": 300, "text":{"string":"Here is a 3D text string!", "perspective":"3d"}
    ,"clearDepthBuffer":true
    ,"scale": [
          {"x":0.4,"y":0.4}
    ]
    ,"color": [
          {"r":0}
    ]
    ,"position": [
         {"x":-5,"y":0,"z":-10}
    ]
    ,"angle": [
           {"x":1,"y":1,"z":1}
          ,{"duration":"0:3", "degreesX":360,"degreesY":360,"degreesZ":360}
    ]
}]);
```

Create sync pattern and apply to animation

```
//Create an inhouse sync pattern
Sync.addSync([
{
    "name":"pattern1"
    ,"start": "0:00", "duration": "0:08"
    ,"pattern": [
         {"start":"0:00","duration":"0:01"}
        ,{"start":"0:02","duration":"0:01"}
        ,{"start":"0:04","duration":"0:01"}
        ,{"start":"0:06","duration":"0:01"}
    ]
}]);
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30"
    ,"layer": 300, "text":{"string":"Here is a 3D text string!", "perspective":"3d"}
    ,"sync":{"name":"pattern1","color":false} //sync pattern applied to everything except color ani
    ,"clearDepthBuffer":true
    ,"scale": [
          {"x":0.6,"y":0.6}
    ]
    ,"color": [
           {"r":0,"a":0}
          ,{"duration":"0:05","a":255}
    ]
    ,"position": [
         {"x":-5,"y":0,"z":-10}
    ]
    ,"angle": [
           {"x":1,"y":1,"z":1}
          ,{"duration":"0:10", "degreesZ":360}
          ,{"duration":"0:10", "degreesY":360}
          ,{"duration":"0:10", "degreesX":360}
    ]
}]);

//Create a GNU Rocket sync pattern
Sync.addSync([
{
    //use GNU Rocket's track "pattern1". Track file should be located here: "data/sync/sync_pattern
    "name":"pattern1", "type":"rocket"
}]);
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30"
    ,"layer": 300, "text":{"string":"Here is a 3D text string!", "perspective":"3d"}
    ,"sync":{"name":"pattern1","color":false} //sync pattern applied to everything except color ani
    ,"clearDepthBuffer":true
    ,"scale": [
          {"x":0.6,"y":0.6}
    ]
    ,"color": [
           {"r":0,"a":0}
          ,{"duration":"0:05","a":255}
    ]
    ,"position": [
         {"x":-5,"y":0,"z":-10}
    ]
    ,"angle": [
           {"x":1,"y":1,"z":1}
          ,{"duration":"0:10", "degreesZ":360}
          ,{"duration":"0:10", "degreesY":360}
          ,{"duration":"0:10", "degreesX":360}
    ]
}]);

//Create a GNU Rocket sync pattern but apply only to one variable in the animation
Sync.addSync([
{
    //use GNU Rocket's track "vinyl.scratch". Track file should be located here: "data/sync/sync_vi
    "name":"vinyl.scratch", "type":"rocket"
}]);
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30"
    ,"layer": 100, "image": "data/vinyl_label.png"
    ,"angle": [
         {"degreesZ":"{return Sync.getSyncValue('vinyl.scratch');}"}
    ]
}
]);
```

3ds file play

```
this.loader.addAnimation([{
     "start": 5.28, "duration": 1.76
    ,"layer": 7, "object": "1.3ds"
    ,"clearDepthBuffer": true
    ,"camera": "Camera01"
    ,"fps": 29
}]);
```

Draw and animate 3D shapes

```
//3D animation works in similar way as with 2D
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30"
    ,"layer": 200, "object": "cube"
    ,"shape":{
        "type":"CUBE"
    },
    "position":[
         {}
        ,{"duration":"0:02","z":-5}
        ,{"duration":"0:01","x":1}
        ,{"duration":"0:01","x":-1}
        ,{"duration":"0:01","y":1}
        ,{"duration":"0:01","y":-1}
        ,{"duration":"0:01","x":0,"y":0}
    ],
    "scale":[
         {}
        ,{"duration":"0:02","x":0.5,"y":2.5}
    ]
    ,"angle":[
         {"x":1}
        ,{"duration":"0:10","degreesX":360,"x":1,"y":1,"z":1}
    ]
    ,"color":[
         {"a":0}
        ,{"duration":"0:01","a":255}
    ]
}]);
```

Setup camera

Look at origo (0,0,0) from z:10:

```
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30", "camera": "cam1", "layer":200
     //where camera is located
     ,"position":[
        {"x":0,"y":0,"z":10}
     ]
     //where camera is looking at
     ,"target":[
        {"x":0,"y":0,"z":0}
     ]
     //camera's up vector
     ,"up":[
        {"x":0,"y":1,"z":0}
     ]
     //camera's perspective setup
     ,"perspective":[
        {"fov":45,"aspect":16/9,"near":1,"far":1000}
     ]
}]);
```

Track a 3D object from distance of z:-5:

```
this.loader.addAnimation([
{
     "start": "0:00", "duration": "0:30", "camera": "cam1", "layer":200
     ,"position":[
        {"x":0,"y":0,"z":-5}
     ]
     ,"target":[
        {"x":0,"y":0,"z":0}
     ]
     //make camera position to track 3d object with name "cube"
     ,"cameraRelativePosition": "cube"
     //make camera target to track 3d object with name "cube"
     ,"cameraRelativeTarget": "cube"
}]);
```

FBO and fullscreen pixel shader example

```
var deformationStart = "1:08";
var deformationDuration = "0:16";
//FBO start
this.loader.addAnimation([
{
     "start": deformationStart, "duration": deformationDuration
    ,"layer": 1
    ,"fbo":{"name":"fbo","action":"begin"}
},
//draw to FBO
{
     "start": deformationStart, "duration": deformationDuration
    ,"layer": 1, "image": "data/jml_fist.png"
}
//FBO end and apply pixel shader
{
     "start": deformationStart, "duration": deformationDuration
    ,"layer": 1
    ,"fbo":{"name":"fbo","action":"end"}
    ,"shader":{"name":"Deformation",
        "variable":[
             {"name":"time","value":["{return getSceneTimeFromStart()/10.0;}"]}
            ,{"name":"effect","type":"int","value":[4]}
        ]
    }
}]);
```

FBO and multipass pixel shader example

```
this.loader.addAnimation([
//render contents rendered in layers 00001 - 00050 to FBO "fbo"
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 1
    ,"fbo":{"name":"fbo","action":"begin"}
},
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 50
    ,"fbo":{"name":"fbo","action":"unbind"}
},
//Render "fbo" via Glow shader (first pass) to "fbo2"
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 51
    ,"fbo":{"name":"fbo2","action":"begin"}
},
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 52
    ,"fbo":{"name":"fbo","action":"draw"}
    ,"shader":{"name":"Glow","variable":[
                 {"name":"direction","value":[0,1]}
                ,{"name":"alpha","value":[1.0]}
            ]}
},
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 53
    ,"fbo":{"name":"fbo2","action":"unbind"}
},
//Draw the original "fbo" to screen
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 54
    ,"image": "fbo.color.fbo"
    ,"color": [
             {"r":255,"g":0,"b":0}
        ]
},
//Draw "fbo2" via Glow shader (second pass) to screen
{
     "start": "0:00", "duration": "0:20"
    ,"layer": 55
    ,"fbo":{"name":"fbo2","action":"draw"}
    ,"shader":{"name":"Glow","variable":[
                 {"name":"direction","value":[1,0]}
                 ,{"name":"alpha","value":[0.4]}
            ]}
}
]);
```

Lighting setup

- Note! There can be maximum 8 lights (indexes 0-7)

Turn light ON:

```
this.loader.addAnimation([
{
     "start": "0:10", "end": "0:58", "light":{"index":0, "action":"begin"}
    ,"layer": 4
    ,"diffuseColor":[
         {"r":255, "g":255, "b":255, "a":255}
    ]
    ,"ambientColor":[
         {"r":"{return colorPalette[colorSpriralColor].r;}", "g":"{return colorPalette[colorSpriral
    ]
    ,"specularColor":[
         {"r":"{return colorPalette[colorSpriralColor].r;}", "g":"{return colorPalette[colorSpriral
    ]
    ,"position":[
         {"x":0.6, "y":0.2, "z":1.0}
    ]
}]);
```

Turn light OFF:

```
this.loader.addAnimation([
{
     "start": "0:10", "end": "0:58", "light":{"index":0, "action":"end"}
    ,"layer": 4
}]);
```

Input handling

In JavaScript it's possible to poll system and input device events, mainly mouse and keyboard.
To enabled input handling set demoScript.useInput as true.

```
while (Input.hasEvents()) //check that there are any available events
{
  var event = Input.pollEvent(); //poll event from the queue

  //check that event type is key up in keyboard
  if (event.type == Input.type.KEYUP)
  {
    //check that "0" is the key where the event occurs
    if (event.keyboard.symbol == Input.Keyboard.symbol.KEY_0)
    {
      //do something...
    }
  }
}
```

Example mouse event:

```
{
  "type": 4, //value corresponds to Input.type
  "time": 0,
  "mouse": {
    "x": 489,
    "y": 1
  }
}
```

Example keyboard event:

```
{
  "type": 2,
  "time": 2.0930001735687256,
  "keyboard": {
    "state": 1, //value corresponds to Input.Keyboard.state
    "symbol": 56 //value corresponds to Input.Keyboard.symbol
  }
}
```

Input constants:

```
Input.type = {
  "NOEVENT": 0,
  "ACTIVEEVENT": 1,
  "KEYDOWN": 2,
  "KEYUP": 3,
  "MOUSEMOTION": 4,
  "MOUSEBUTTONDOWN": 5,
  "MOUSEBUTTONUP": 6,
  "JOYAXISMOTION": 7,
  "JOYBALLMOTION": 8,
  "JOYHATMOTION": 9,
  "JOYBUTTONDOWN": 10,
  "JOYBUTTONUP": 11,
  "QUIT": 12,
  "SYSWMEVENT": 13,
  "EVENT_RESERVEDA": 14,
  "EVENT_RESERVEDB": 15,
  "VIDEORESIZE": 16,
  "VIDEOEXPOSE": 17,
  "USEREVENT": 24
};

Input.Keyboard = {
  "state": {
    "RELEASED": 0,
    "PRESSED": 1
  },
  "symbol": {
    "KEY_UNKNOWN": 0,
    "KEY_FIRST": 0,
    "KEY_BACKSPACE": 8,
    "KEY_TAB": 9,
    "KEY_CLEAR": 12,
    "KEY_RETURN": 13,
    "KEY_PAUSE": 19,
    "KEY_ESCAPE": 27,
    "KEY_SPACE": 32,
    "KEY_EXCLAIM": 33,
    "KEY_QUOTEDBL": 34,
    "KEY_HASH": 35,
    "KEY_DOLLAR": 36,
    "KEY_AMPERSAND": 38,
    "KEY_QUOTE": 39,
    "KEY_LEFTPAREN": 40,
    "KEY_RIGHTPAREN": 41,
    "KEY_ASTERISK": 42,
    "KEY_PLUS": 43,
    "KEY_COMMA": 44,
    "KEY_MINUS": 45,
    "KEY_PERIOD": 46,
    "KEY_SLASH": 47,
    "KEY_0": 48,
    "KEY_1": 49,
    "KEY_2": 50,
    "KEY_3": 51,
    "KEY_4": 52,
    "KEY_5": 53,
    "KEY_6": 54,
    "KEY_7": 55,
    "KEY_8": 56,
    "KEY_9": 57,
    "KEY_COLON": 58,
    "KEY_SEMICOLON": 59,
    "KEY_LESS": 60,
    "KEY_EQUALS": 61,
    "KEY_GREATER": 62,
    "KEY_QUESTION": 63,
    "KEY_AT": 64,
    "KEY_LEFTBRACKET": 91,
    "KEY_BACKSLASH": 92,
    "KEY_RIGHTBRACKET": 93,
    "KEY_CARET": 94,
    "KEY_UNDERSCORE": 95,
    "KEY_BACKQUOTE": 96,
    "KEY_a": 97,
    "KEY_b": 98,
    "KEY_c": 99,
    "KEY_d": 100,
    "KEY_e": 101,
    "KEY_f": 102,
    "KEY_g": 103,
    "KEY_h": 104,
    "KEY_i": 105,
    "KEY_j": 106,
    "KEY_k": 107,
    "KEY_l": 108,
    "KEY_m": 109,
    "KEY_n": 110,
    "KEY_o": 111,
    "KEY_p": 112,
    "KEY_q": 113,
    "KEY_r": 114,
    "KEY_s": 115,
    "KEY_t": 116,
    "KEY_u": 117,
    "KEY_v": 118,
    "KEY_w": 119,
    "KEY_x": 120,
    "KEY_y": 121,
    "KEY_z": 122,
    "KEY_DELETE": 127,
    "KEY_WORLD_0": 160,
    "KEY_WORLD_1": 161,
    "KEY_WORLD_2": 162,
    "KEY_WORLD_3": 163,
    "KEY_WORLD_4": 164,
    "KEY_WORLD_5": 165,
    "KEY_WORLD_6": 166,
    "KEY_WORLD_7": 167,
    "KEY_WORLD_8": 168,
    "KEY_WORLD_9": 169,
    "KEY_WORLD_10": 170,
    "KEY_WORLD_11": 171,
    "KEY_WORLD_12": 172,
    "KEY_WORLD_13": 173,
    "KEY_WORLD_14": 174,
    "KEY_WORLD_15": 175,
    "KEY_WORLD_16": 176,
    "KEY_WORLD_17": 177,
    "KEY_WORLD_18": 178,
    "KEY_WORLD_19": 179,
    "KEY_WORLD_20": 180,
    "KEY_WORLD_21": 181,
    "KEY_WORLD_22": 182,
    "KEY_WORLD_23": 183,
    "KEY_WORLD_24": 184,
    "KEY_WORLD_25": 185,
    "KEY_WORLD_26": 186,
    "KEY_WORLD_27": 187,
    "KEY_WORLD_28": 188,
    "KEY_WORLD_29": 189,
    "KEY_WORLD_30": 190,
    "KEY_WORLD_31": 191,
    "KEY_WORLD_32": 192,
    "KEY_WORLD_33": 193,
    "KEY_WORLD_34": 194,
    "KEY_WORLD_35": 195,
    "KEY_WORLD_36": 196,
    "KEY_WORLD_37": 197,
    "KEY_WORLD_38": 198,
    "KEY_WORLD_39": 199,
    "KEY_WORLD_40": 200,
    "KEY_WORLD_41": 201,
    "KEY_WORLD_42": 202,
    "KEY_WORLD_43": 203,
    "KEY_WORLD_44": 204,
    "KEY_WORLD_45": 205,
    "KEY_WORLD_46": 206,
    "KEY_WORLD_47": 207,
    "KEY_WORLD_48": 208,
    "KEY_WORLD_49": 209,
    "KEY_WORLD_50": 210,
    "KEY_WORLD_51": 211,
    "KEY_WORLD_52": 212,
    "KEY_WORLD_53": 213,
    "KEY_WORLD_54": 214,
    "KEY_WORLD_55": 215,
    "KEY_WORLD_56": 216,
    "KEY_WORLD_57": 217,
    "KEY_WORLD_58": 218,
    "KEY_WORLD_59": 219,
    "KEY_WORLD_60": 220,
    "KEY_WORLD_61": 221,
    "KEY_WORLD_62": 222,
    "KEY_WORLD_63": 223,
    "KEY_WORLD_64": 224,
    "KEY_WORLD_65": 225,
    "KEY_WORLD_66": 226,
    "KEY_WORLD_67": 227,
    "KEY_WORLD_68": 228,
    "KEY_WORLD_69": 229,
    "KEY_WORLD_70": 230,
    "KEY_WORLD_71": 231,
    "KEY_WORLD_72": 232,
    "KEY_WORLD_73": 233,
    "KEY_WORLD_74": 234,
    "KEY_WORLD_75": 235,
    "KEY_WORLD_76": 236,
    "KEY_WORLD_77": 237,
    "KEY_WORLD_78": 238,
    "KEY_WORLD_79": 239,
    "KEY_WORLD_80": 240,
    "KEY_WORLD_81": 241,
    "KEY_WORLD_82": 242,
    "KEY_WORLD_83": 243,
    "KEY_WORLD_84": 244,
    "KEY_WORLD_85": 245,
    "KEY_WORLD_86": 246,
    "KEY_WORLD_87": 247,
    "KEY_WORLD_88": 248,
    "KEY_WORLD_89": 249,
    "KEY_WORLD_90": 250,
    "KEY_WORLD_91": 251,
    "KEY_WORLD_92": 252,
    "KEY_WORLD_93": 253,
    "KEY_WORLD_94": 254,
    "KEY_WORLD_95": 255,
    "KEY_KP0": 256,
    "KEY_KP1": 257,
    "KEY_KP2": 258,
    "KEY_KP3": 259,
    "KEY_KP4": 260,
    "KEY_KP5": 261,
    "KEY_KP6": 262,
    "KEY_KP7": 263,
    "KEY_KP8": 264,
    "KEY_KP9": 265,
    "KEY_KP_PERIOD": 266,
    "KEY_KP_DIVIDE": 267,
    "KEY_KP_MULTIPLY": 268,
    "KEY_KP_MINUS": 269,
    "KEY_KP_PLUS": 270,
    "KEY_KP_ENTER": 271,
    "KEY_KP_EQUALS": 272,
    "KEY_UP": 273,
    "KEY_DOWN": 274,
    "KEY_RIGHT": 275,
    "KEY_LEFT": 276,
    "KEY_INSERT": 277,
    "KEY_HOME": 278,
    "KEY_END": 279,
    "KEY_PAGEUP": 280,
    "KEY_PAGEDOWN": 281,
    "KEY_F1": 282,
    "KEY_F2": 283,
    "KEY_F3": 284,
    "KEY_F4": 285,
    "KEY_F5": 286,
    "KEY_F6": 287,
    "KEY_F7": 288,
    "KEY_F8": 289,
    "KEY_F9": 290,
    "KEY_F10": 291,
    "KEY_F11": 292,
    "KEY_F12": 293,
    "KEY_F13": 294,
    "KEY_F14": 295,
    "KEY_F15": 296,
    "KEY_NUMLOCK": 300,
    "KEY_CAPSLOCK": 301,
    "KEY_SCROLLOCK": 302,
    "KEY_RSHIFT": 303,
    "KEY_LSHIFT": 304,
    "KEY_RCTRL": 305,
    "KEY_LCTRL": 306,
    "KEY_RALT": 307,
    "KEY_LALT": 308,
    "KEY_RMETA": 309,
    "KEY_LMETA": 310,
    "KEY_LSUPER": 311,
    "KEY_RSUPER": 312,
    "KEY_MODE": 313,
    "KEY_COMPOSE": 314,
    "KEY_HELP": 315,
    "KEY_PRINT": 316,
    "KEY_SYSREQ": 317,
    "KEY_BREAK": 318,
    "KEY_MENU": 319,
    "KEY_POWER": 320,
    "KEY_EURO": 321,
    "KEY_UNDO": 322
  }
};
```

Custom JavaScript functions

Simple example

Definition in the init method:

```
    this.loader.addAnimation([
    {
         "start": "0:10", "end": "0:58"
        ,"layer": 4
        ,"initFunction":"{initFunction(animation);}"
        ,"runFunction":"{drawFunction(animation);}"
    }]);
```

Custom JavaScript 3d object handling function

Definition in the init method:

```
this.loader.addAnimation([
{
     "start": "0:10", "end": "0:58", "object":"disco","shape":{"type":"CUSTOM"}
    ,"layer": 4
    ,"initFunction":"{initDiscoball(animation);}"
    ,"objectFunction":"{drawDiscoball(animation);}"
    ,"position":[
         {"z":-5}
        ,{"start":"0:55","duration":"0:03","y":4}
    ]
    ,"angle":[
        {"degreesX":90, "degreesZ":"{return getSceneTimeFromStart()*30;}"}
    ]
}]);
```

Actual code:

```
var discoball;
function initDiscoball(animation)
{
    discoball = setObjectSphereData("discoball", 1,30,15);
}

function drawDiscoball(animation)
{
    var time = getSceneTimeFromStart();

    glDisable(GL_TEXTURE_2D);
    glPushMatrix();
    glShadeModel(GL_FLAT);

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    setObjectColor(discoball.ptr, 1,1,1,1);
    drawObject(discoball.ptr);
    glEnable(GL_BLEND);
    glColor4f(1,1,1,1);

    glLineWidth(3);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    var col = 0.25;
    setObjectColor(discoball.ptr, col,col,col,0.15);
    drawObject(discoball.ptr);
    glLineWidth(1);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glShadeModel(GL_SMOOTH);
    glPopMatrix();
}
```

Particle animation

Definition in the init method:

```
this.loader.addAnimation([
{
     "start": "0:40.5", "end": "1:11"
    ,"layer": 3
    ,"initFunction":"{initStarParticles(animation);}"
    ,"runFunction":"{drawStarParticles(animation);}"
}]);
```

Actual code:

```
var particleContainer; //variable that will store the particle data

//particleInit will be called every time particle needs to be initialized
function particleInit(particleContainer, particle)
{
    //set start and end angle degree of the single particle
    setParticleAngleRange(particle.ptr, 0, 0, random()*360, 0, 0, random()*360*(random()*2-1));

    //setup start and end position of the single particle
    var starW = 615; //texture width
    var starH = 629; //texture height
    var x = getScreenWidth()/2-(starW*scale)/2;
    var y = getScreenHeight()/2-(starH*scale)/2;
    var precision = getScreenHeight();
    var radius = getScreenWidth();
    var angle = (random()*precision+precision)*2*Math.PI/precision;
    var endX = Math.cos(angle) * radius;
    var endY = Math.sin(angle) * radius;
    setParticlePositionRange(particle.ptr, x, y, 0, endX, endY, 0);

    //setup the lifespan of the single particle
    var lifeSeconds = 3;
    var startTime = getSceneTimeFromStart()+random()*lifeSeconds*2;
    var duration = lifeSeconds+random()*lifeSeconds;
    setParticleTime(particle.ptr, startTime, duration);
}

//do default setup and bindings of the particles
function initStarParticles(animation)
{
    particleContainer = initParticleContainer();
    setParticleContainerTime(particleContainer.ptr, animation.start, animation.duration);

    //set perspective to 2D. default is 3D
    setParticleContainerPerspective3d(particleContainer.ptr, 0);

    //bind function particleInit so that it will be called every time particle needs to be initiali
    bindParticleContainerInitParticleFunction(particleContainer.ptr, "particleInit");

    //by default during particle init a texture from the particle container default texture list wi
    setParticleContainerDefaultTextureList(particleContainer.ptr,
        [
             imageLoadImage("data/star_open.png").ptr
            ,imageLoadImage("data/star.png").ptr
        ]
    );

    //initialize 100 particles
    initParticleContainerParticles(particleContainer.ptr, 0, 100);
}

//do the calculation and drawing of the particles
function drawStarParticles(animation)
{
    drawParticleContainer(particleContainer.ptr);
}
```

Bindings to JavaScript

OpenGL

- OpenGL <= 1.2 extensively supported. Some missing functions here and there.
- OpenGL reference pages

Engine function bindings

- WARNING! Bindings and naming conventions of the functions are subject to change. Please ensure enough abstraction so that porting and upgrading would be as easy as possible.

Sound

- void soundLoadPlaylist(const char *_filename);
- int soundAddSongToPlaylist(const char *_filename, const char *title, int length);
- void soundLoadSong(int song_number);
- double soundGetSongCurrentPlayTime(void);
- void soundPlaySong(int song_number);
- void soundPause(void);
- void soundStop(void);
- void soundMute(int _mute_sound);
- int soundIsMute(void);
- void soundPreviousTrack(void);
- void soundNextTrack(void);
- int soundGetPlaylistSize(void);
- int soundGetCurrentSong(void);
- int soundGetSongLength(int song_number);
- const char* soundGetSongFilename(int song_number);
- const char* soundGetSongName(int song_number);
- int soundGetTrackNumber(int song_number);
- int soundIsPlaying(void);
- void soundClearPlaylist(void);
- void soundSetPosition(double position);

Scene and time

- const char* getSceneName();
- double getSceneStartTime();
- double getSceneEndTime();
- double getSceneTimeFromStart();
- double getSceneProgressPercent();
- double convertTimeToSeconds(const char *time);
- double timerGetBeatsPerMinute(void);
- void timerSetBeatsPerMinute(double bpm);
- double timerGetBeatInSeconds(void);
- double timerGetCurrentBeat(void);
- double timerGetTime(void);
- void timerInit(double newEndTime);
- void timerSleep(int millis);

- double timerGetFpsCorrection(void);
- void syncEditorSetRowsPerBeat(int _rowsPerBeat);
- int syncEditorGetRowsPerBeat(void);
- void* syncEditorGetTrack(const char *trackName);
- double syncEditorGetTrackCurrentValue(void *trackPointer);

Screen and window

- int getScreenWidth(void);
- int getScreenHeight(void);
- double getWindowScreenAreaAspectRatio();
- void setWindowScreenAreaAspectRatio(double width, double height);
- void setScreenDimensions(int height, int width);
- void windowSetTitle(const char *newTitle);
- void viewReset(void);

Camera

- void setCameraPositionObject(object3d_t *object);
- void setCameraTargetObject(object3d_t *object);
- void setCameraPerspective(double fovy, double aspect, double zNear, double zFar);
- void setCameraPosition(float x, float y, float z);
- void setCameraLookAt(float x, float y, float z);
- void setCameraUpVector(float x, float y, float z);
- camera_t* getCamera();

Lighting

- void setLight4f(unsigned int light, unsigned int type, float f1, float f2, float f3, float f4);
- void setLight4ub(unsigned int light, unsigned int type, unsigned int f1, unsigned int f2, unsigned int f3, unsigned int f4);
- void lightSetAmbientColor(unsigned int i, float r, float g, float b, float a);
- void lightSetDiffuseColor(unsigned int i, float r, float g, float b, float a);
- void lightSetSpecularColor(unsigned int i, float r, float g, float b, float a);
- void lightSetPosition(unsigned int i, float x, float y, float z);
- void lightSetPositionObject(unsigned int i, object3d_t *positionObject);
- void lightInit(unsigned int i);
- void lightSetOn(unsigned int i);
- void lightSetOff(unsigned int i);
- int isLightingEnabled();

Text

- void setTextPivot(double x, double y, double z);
- void setTextRotation(double x, double y, double z);
- void setTextSize(double w, double h);
- void setTextDefaults(void);
- void setTextPosition(double x, double y, double z);
- void alignTextCenter(int center);
- void setDrawTextString(const char *txt);
- int getTextStringWidth();
- int getTextStringHeight();
- void drawText2d();
- void drawText3d();

Input

- void jsSetUseInput(int _useInput);
- int jsIsUseInput(void);

Graphics

- void setClearColor(float r, float g, float b, float a);
- void perspective2dBegin(int w, int h);
- void perspective2dEnd(void);

Frame buffer objects

- void fboBind(fbo_t* fbo);
- fbo_t* fboInit(const char *name);
- void fboDeinit(fbo_t* fbo);
- void fboStoreDepth(fbo_t* fbo, int _storeDepth);
- int fboGenerateFramebuffer(fbo_t* fbo);
- void fboSetDimensions(fbo_t* fbo, unsigned int width, unsigned int height);
- void fboSetRenderDimensions(fbo_t* fbo, double widthPercent, double heightPercent);
- int fboGetWidth(fbo_t* fbo);
- int fboGetHeight(fbo_t* fbo);
- void fboUpdateViewport(fbo_t* fbo);
- void fboBindTextures(fbo_t* fbo);

Shaders

- unsigned int getUniformLocation(const char *variable);
- void glUniformf(unsigned int uniformLocation, float value...);
- void glUniformi(unsigned int uniformLocation, int value...);
- void disableShaderProgram();
- void activateShaderProgram(const char *name);

2D image handling

- texture_t* imageLoadImage(const char* filename);
- void setTexturePerspective3d(texture_t *texture, int perspective3d);
- void setTextureBlendFunc(texture_t *texture, unsigned int srcBlend, unsigned int dstBlend);
- void setTextureCanvasDimensions(texture_t *texture, int w, int h);
- void setTextureUvDimensions(texture_t *texture, double uMin, double vMin, double uMax, double vMax);
- void setTextureSizeToScreenSize(texture_t *texture);
- void setTextureCenterAlignment(texture_t *texture, int center);
- void setTexturePosition(texture_t *texture, double x, double y, double z);
- void setTexturePivot(texture_t *texture, double x, double y, double z);
- void setTextureScale(texture_t *texture, double scaleW, double scaleH);
- void setTextureRotation(texture_t* texture, double degreesX, double degreesY, double degreesZ, double x, double y, double z);
- void setTextureUnitTexture(texture_t *texture, unsigned int unitIndex, texture_t *textureDst);
- void setTextureDefaults(texture_t *texture);
- void drawTexture(texture_t *texture);

2D video handling

- video_t *videoLoad(const char *filename);
- void videoSetSpeed(video_t *video, double speed);
- void videoSetFps(video_t *video, double fps);
- void videoSetLoop(video_t *video, int loop);
- void videoPlay(video_t *video);
- void videoSetStartTime(video_t *video, float startTime);
- void videoSetTime(video_t *video, float time);
- void videoStop(video_t *video);
- void videoPause(video_t *video);
- void videoDraw(video_t *video);

3D object handling

- object3d_t* getObjectFromMemory(const char *filename);
- void useObjectLighting(object3d_t* object, int useObjectLighting);
- void useObjectCamera(object3d_t* object, int useObjectCamera);
- void useObjectNormals(object3d_t* object, int useObjectNormals);
- void useObjectTextureCoordinates(object3d_t* object, int useObjectTextureCoordinates);
- void setObjectScale(object3d_t* object, float x, float y, float z);
- void setObjectPosition(object3d_t* object, float x, float y, float z);
- void setObjectPivot(object3d_t* object, float x, float y, float z);
- void setObjectRotation(object3d_t* object, float degreesX, float degreesY, float degreesZ, float x, float y, float z);
- void setObjectColor(object3d_t* object, float r, float g, float b, float a);
- object3d_t* setObjectDiskData(const char *name, object3d_t *object, double inner, double outer, int slices, int loops);
- object3d_t* setObjectCylinderData(const char *name, object3d_t *object, double base, double top, double height, int slices, int stacks);
- object3d_t* setObjectSphereData(const char *name, object3d_t *object, double radius, int lats, int longs);
- void drawObject(void* object_ptr, const char* displayCamera, double displayFrame, int clear);
- void* loadObjectBasicShape(const char * name, int objectType);
- void *loadObject(const char * filename);
- int replaceObjectTexture(object3d_t *object, const char *findTextureName, const char *replaceTextureName);

Particles

- void deinitParticleContainer(void *particleContainerPointer);
- particleContainer_t* initParticleContainer(particleContainer_t *particleContainer);
- void initParticleContainerParticles(particleContainer_t *particleContainer, unsigned int particleI, unsigned int count);
- void drawParticleContainer(particleContainer_t *particleContainer);
- unsigned int getParticleContainerParticleCount(particleContainer_t *particleContainer);
- void setParticleContainerPerspective3d(particleContainer_t *particleContainer, int perspective3d);
- void setParticleContainerDefaultTextureList(particleContainer_t *particleContainer, texture_t **particleDefaultTextureList, unsigned int particleDefaultTextureCount);
- void setParticleContainerTime(particleContainer_t *particleContainer, float startTime, float duration);
- void setParticleContainerParticleDurationRange(particleContainer_t *particleContainer, float particleDurationMin, float particleDurationMax);
- void setParticleContainerParticleFadeTimeRange(particleContainer_t *particleContainer, float particleFadeInTime, float particleFadeOutTime);
- void setParticleContainerParticleInitDelay(particleContainer_t *particleContainer, float particleInitDelay);
- void setParticleContainerParticleInitCountMax(particleContainer_t *particleContainer, int particleInitCountMax);
- void setParticleContainerPosition(particleContainer_t *particleContainer, point3d_t position);
- void setParticleContainerPositionRange(particleContainer_t *particleContainer, point3d_t positionMin, point3d_t positionMax);
- void setParticleContainerParticleScaleRange(particleContainer_t *particleContainer, point3d_t particleScaleMin, point3d_t particleScaleMax);
- void setParticleContainerParticleAngleRange(particleContainer_t *particleContainer, point3d_t particleAngleMin, point3d_t particleAngleMax);
- void setParticleContainerParticleColor(particleContainer_t *particleContainer, color_t particleColor);
- void setParticleContainerParticlePivot(particleContainer_t *particleContainer, point3d_t particlePivot);
- void setParticleContainerDirection(particleContainer_t *particleContainer, point3d_t direction);
- void setParticleTexture(particle_t *particle, texture_t *texture);
- void setParticleActive(particle_t *particle, int active);
- float getParticleProgress(particle_t *particle);
- void setParticleTime(particle_t *particle, float startTime, float duration);
- void setParticlePosition(particle_t *particle, point3d_t position);
- void setParticlePositionRange(particle_t *particle, point3d_t startPosition, point3d_t endPosition);
- void setParticleScale(particle_t *particle, point3d_t scale);
- void setParticleScaleRange(particle_t *particle, point3d_t startScale, point3d_t endScale);
- void setParticleAngle(particle_t *particle, point3d_t angle);
- void setParticleAngleRange(particle_t *particle, point3d_t startAngle, point3d_t endAngle);
- void setParticlePivot(particle_t *particle, point3d_t pivot);
- void setParticleColor(particle_t *particle, color_t color);

Math

- double interpolateLinear(percent, startValue, endValue);
- double interpolateSmoothStep(percent, startValue, endValue);
- double interpolateSmootherStep(percent, startValue, endValue);
- double interpolate(percent, startValue, endValue, int type);
    - types
        - 0: linear interpolation
        - 1: smooth step interpolation
        - 2: smoother step interpolation
- double random();
- double random(int seedValue);

Miscellaneous

- void screenPrint(const char *text);
- void debugPrint(const char *text...);
- void debugWarningPrint(const char *text...);
- void debugErrorPrint(const char *text...);
- const char *readFile(const char *filename);
- void evalFile(const char *filename...);

Engine internal

- void setPlaylistMusic(const char *file);
- void setPlaylistLength(const char *length);
- void setResourceCount(int _sceneResourceCount);
- playerScene *addPlayerScene(playerScene *parentScene, const char *name, const char *effectName, const char *startString, const char *durationString);
- playerEffect *addPlayerEffect(const char *name, const char *reference);
- void addTwVariable(const char* sceneName, const char* variableName, const char* variableType, const char* variableDefinition);
- shaderProgram_t *shaderProgramLoad(const char *name);
- void shaderProgramAttachAndLink(shaderProgram_t *shaderProgram);
- void shaderProgramAddShaderByName(const char *shaderProgramName, const char *shaderName);
- shaderProgram_t* getShaderProgramFromMemory(const char *name);
- void shaderProgramUse(shaderProgram_t *shaderProgram);
- void setMenuComponentSelected(int componentIndex);
- void setMenuResolution(int componentIndex, const char *resolutionString);
- int isUserExit(void);

Copyrights and licensing

- Matrix.js / Vector.js from lightgl.js