

A simple and easy-to-use library to enjoy videogames programming

[raylib Discord server][github.com/raysans/raylib][raylib.h]

v5.0 quick reference card [download as PDF]

Chinese Translation: [以下为raylib所有API接口中文释义](#)

module: core →

```
// Window-related functions
void InitWindow(int width, int height, const char *title); // Initialize window and OpenGL context
void CloseWindow(void); // Close window and unload OpenGL context
bool WindowShouldClose(void); // Check if application should close (KEY_ESCAPE pressed or windows close icon clicked)
bool IsWindowReady(void); // Check if window has been initialized successfully
bool IsWindowFullscreen(void); // Check if window is currently fullscreen
bool IsWindowHidden(void); // Check if window is currently hidden (only PLATFORM_DESKTOP)
bool IsWindowMinimized(void); // Check if window is currently minimized (only PLATFORM_DESKTOP)
bool IsWindowMaximized(void); // Check if window is currently maximized (only PLATFORM_DESKTOP)
bool IsWindowFocused(void); // Check if window is currently focused (only PLATFORM_DESKTOP)
bool IsWindowResized(void); // Check if window has been resized last frame
bool IsWindowState(unsigned int flag); // Check if one specific window flag is enabled
void SetWindowState(unsigned int flags); // Set window configuration state using flags (only PLATFORM_DESKTOP)
void ClearWindowState(unsigned int flags); // Clear window configuration state flags
void ToggleFullscreen(void); // Toggle window state: fullscreen/windowed (only PLATFORM_DESKTOP)
void ToggleBorderlessWindowed(void); // Toggle window state: borderless windowed (only PLATFORM_DESKTOP)
void MaximizeWindow(void); // Set window state: maximized, if resizable (only PLATFORM_DESKTOP)
void MinimizeWindow(void); // Set window state: minimized, if resizable (only PLATFORM_DESKTOP)
void RestoreWindow(void); // Set window state: not minimized/maximized (only PLATFORM_DESKTOP)
void SetWindowIcon(Image image); // Set icon for window (single image, RGBA 32bit, only PLATFORM_DESKTOP)
void SetWindowIcons(Image *images, int count); // Set icon for window (multiple images, RGBA 32bit, only PLATFORM_DESKTOP)
void SetWindowTitle(const char *title); // Set title for window (only PLATFORM_DESKTOP and PLATFORM_WEB)
void SetWindowPosition(int x, int y); // Set window position on screen (only PLATFORM_DESKTOP)
void SetWindowMonitor(int monitor); // Set monitor for the current window
void SetWindowMinSize(int width, int height); // Set window minimum dimensions (for FLAG_WINDOW_RESIZABLE)
void SetWindowMaxSize(int width, int height); // Set window maximum dimensions (for FLAG_WINDOW_RESIZABLE)
void SetWindowSize(int width, int height); // Set window dimensions
void SetWindowOpacity(float opacity); // Set window opacity [0.0f..1.0f] (only PLATFORM_DESKTOP)
void SetWindowFocused(void); // Set window focused (only PLATFORM_DESKTOP)
void *GetWindowHandle(void); // Get native window handle
int GetScreenWidth(void); // Get current screen width
int GetScreenHeight(void); // Get current screen height
int GetRenderWidth(void); // Get current render width (it considers HiDPI)
int GetRenderHeight(void); // Get current render height (it considers HiDPI)
int GetMonitorCount(void); // Get number of connected monitors
int GetCurrentMonitor(void); // Get current connected monitor
Vector2 GetMonitorPosition(int monitor); // Get specified monitor position
int GetMonitorWidth(int monitor); // Get specified monitor width (current video mode used by monitor)
int GetMonitorHeight(int monitor); // Get specified monitor height (current video mode used by monitor)
int GetMonitorPhysicalWidth(int monitor); // Get specified monitor physical width in millimetres
int GetMonitorPhysicalHeight(int monitor); // Get specified monitor physical height in millimetres
int GetMonitorRefreshRate(int monitor); // Get specified monitor refresh rate
Vector2 GetWindowPosition(void); // Get window position XY on monitor
Vector2 GetWindowSizeDPI(void); // Get window scale DPI factor
const char *GetMonitorName(int monitor); // Get the human-readable, UTF-8 encoded name of the specified monitor
void SetClipboardText(const char *text); // Set clipboard text content
const char *GetClipboardText(void); // Get clipboard text content
void EnableEventWaiting(void); // Enable waiting for events on EndDrawing(), no automatic event polling
void DisableEventWaiting(void); // Disable waiting for events on EndDrawing(), automatic events polling

// Cursor-related functions
void ShowCursor(void); // Shows cursor
void HideCursor(void); // Hides cursor
bool IsCursorHidden(void); // Check if cursor is not visible
void EnableCursor(void); // Enables cursor (unlock cursor)
void DisableCursor(void); // Disables cursor (lock cursor)
bool IsCursorOnScreen(void); // Check if cursor is on the screen

// Drawing-related functions
void ClearBackground(Color color); // Set background color (framebuffer clear color)
void BeginDrawing(void); // Setup canvas (framebuffer) to start drawing
void EndDrawing(void); // End canvas drawing and swap buffers (double buffering)
void BeginMode2D(Camera2D camera); // Begin 2D mode with custom camera (2D)
void EndMode2D(void); // Ends 2D mode with custom camera
void BeginMode3D(Camera3D camera); // Begin 3D mode with custom camera (3D)
void EndMode3D(void); // Ends 3D mode and returns to default 2D orthographic mode
void BeginTextureMode(RenderTexture2D target); // Begin drawing to render texture
void EndTextureMode(void); // Ends drawing to render texture
void BeginShaderMode(Shader shader); // Begin custom shader drawing
void EndShaderMode(void); // End custom shader drawing (use default shader)
void BeginBlendMode(int mode); // Begin blending mode (alpha, additive, multiplied, subtract, custom)
void EndBlendMode(void); // End blending mode (reset to default: alpha blending)
void BeginScissorMode(int x, int y, int width, int height); // Begin scissor mode (define screen area for following drawing)
void EndScissorMode(void); // End scissor mode
void BeginVrStereoMode(VrStereoConfig config); // Begin stereo rendering (requires VR simulator)
void EndVrStereoMode(void); // End stereo rendering (requires VR simulator)

// VR stereo config functions for VR simulator
VrStereoConfig LoadVrStereoConfig(VrDeviceInfo device); // Load VR stereo config for VR simulator device parameters
void UnloadVrStereoConfig(VrStereoConfig config); // Unload VR stereo config

// Shader management functions
// NOTE: Shader functionality is not available on OpenGL 1.1
Shader LoadShader(const char *vsFileName, const char *fsFileName); // Load shader from files and bind default locations
Shader LoadShaderFromMemory(const char *vsCode, const char *fsCode); // Load shader from code strings and bind default locations
bool IsShaderReady(Shader shader); // Check if a shader is ready
int GetShaderLocation(Shader shader, const char *uniformName); // Get shader uniform location
int GetShaderLocationAttrib(Shader shader, const char *attribName); // Get shader attribute location
void SetShaderValue(Shader shader, int locIndex, const void *value, int uniformType); // Set shader uniform value
void SetShaderValueV(Shader shader, int locIndex, const void *value, int uniformType, int count); // Set shader uniform value vector
void SetShaderValueMatrix(Shader shader, int locIndex, Matrix mat); // Set shader uniform value (matrix 4x4)
void SetShaderValueTexture(Shader shader, int locIndex, Texture2D texture); // Set shader uniform value for texture (sampler2d)
void UnloadShader(Shader shader); // Unload shader from GPU memory (VRAM)
```

```

// Screen-space-related functions
Ray GetMouseRay(Vector2 mousePosition, Camera camera);           // Get a ray trace from mouse position
Matrix GetCameraMatrix(Camera camera);                          // Get camera transform matrix (view matrix)
Matrix GetCameraMatrix2D(Camera2D camera);                     // Get camera 2d transform matrix
Vector2 GetWorldToScreen(Vector3 position, Camera camera);    // Get the screen space position for a 3d world space position
Vector2 GetScreenToWorld2D(Vector2 position, Camera2D camera); // Get the world space position for a 2d camera screen space position
Vector2 GetWorldToScreenEx(Vector3 position, Camera camera, int width, int height); // Get size position for a 3d world space position
Vector2 GetWorldToScreen2D(Vector2 position, Camera2D camera); // Get the screen space position for a 2d camera world space position

// Timing-related functions
void SetTargetFPS(int fps);                                     // Set target FPS (maximum)
float GetFrameTime(void);                                     // Get time in seconds for last frame drawn (delta time)
double GetTime(void);                                         // Get elapsed time in seconds since InitWindow()
int GetFPS(void);                                            // Get current FPS

// Custom frame control functions
// NOTE: Those functions are intended for advance users that want full control over the frame processing
// By default EndDrawing() does this job: draws everything + SwapScreenBuffer() + manage frame timing + PollInputEvents()
// To avoid that behaviour and control frame processes manually, enable in config.h: SUPPORT_CUSTOM_FRAME_CONTROL
void SwapScreenBuffer(void);                                    // Swap back buffer with front buffer (screen drawing)
void PollInputEvents(void);                                    // Register all input events
void WaitTime(double seconds);                                // Wait for some time (halt program execution)

// Random values generation functions
void SetRandomSeed(unsigned int seed);                         // Set the seed for the random number generator
int GetRandomValue(int min, int max);                          // Get a random value between min and max (both included)
int *LoadRandomSequence(unsigned int count, int min, int max); // Load random values sequence, no values repeated
void UnloadRandomSequence(int *sequence);                      // Unload random values sequence

// Misc. functions
void TakeScreenshot(const char *fileName);                   // Takes a screenshot of current screen (filename extension defines format)
void SetConfigFlags(unsigned int flags);                      // Setup init configuration flags (view FLAGS)
void OpenURL(const char *url);                               // Open URL with default system browser (if available)

// NOTE: Following functions implemented in module [utils]
//-----
void TraceLog(int logLevel, const char *text, ...);          // Show trace log messages (LOG_DEBUG, LOG_INFO, LOG_WARNING, LOG_ERROR...)
void SetTraceLogLevel(int logLevel);                          // Set the current threshold (minimum) log level
void *MemAlloc(unsigned int size);                           // Internal memory allocator
void *MemRealloc(void *ptr, unsigned int size);              // Internal memory reallocator
void MemFree(void *ptr);                                    // Internal memory free

// Set custom callbacks
// WARNING: Callbacks setup is intended for advance users
void SetTraceLogCallback(TraceLogCallback callback);        // Set custom trace log
void SetLoadFileDataCallback(LoadFileDataCallback callback); // Set custom file binary data loader
void SetSaveFileDataCallback(SaveFileDataCallback callback); // Set custom file binary data saver
void SetLoadFileTextCallback(LoadFileTextCallback callback); // Set custom file text data loader
void SetSaveFileTextCallback(SaveFileTextCallback callback); // Set custom file text data saver

// Files management functions
unsigned char *LoadFileData(const char *fileName, int *dataSize); // Load file data as byte array (read)
void UnloadFileData(unsigned char *data);                      // Unload file data allocated by LoadFileData()
bool SaveFileData(const char *fileName, void *data, int dataSize); // Save data to file from byte array (write), returns true on success
bool ExportDataAsCode(const unsigned char *data, int dataSize, const char *fileName); // Export data to code (.h), returns true on success
char *LoadFileText(const char *fileName);                      // Load text data from file (read), returns a '\0' terminated string
void UnloadFileText(char *text);                             // Unload file text data allocated by LoadFileText()
bool SaveFileText(const char *fileName, char *text);          // Save text data to file (write), string must be '\0' terminated, returns true
//-----

// File system functions
bool FileExists(const char *fileName);                         // Check if file exists
bool DirectoryExists(const char *dirPath);                     // Check if a directory path exists
bool IsFileExtension(const char *fileName, const char *ext);  // Check file extension (including point: .png, .wav)
int GetFileLength(const char *fileName);                      // Get file length in bytes (NOTE: GetFileSize() conflicts with windows.h)
const char *GetFileExtension(const char *fileName);           // Get pointer to extension for a given filename string (includes dot: '.png')
const char *GetFileName(const char *fileName);                // Get pointer to filename for a path string
const char *GetFileNameWithoutExt(const char *filePath);     // Get filename string without extension (uses static string)
const char *GetDirectoryPath(const char *filePath);          // Get full path for a given fileName with path (uses static string)
const char *GetPrevDirectoryPath(const char *dirPath);        // Get previous directory path for a given path (uses static string)
const char *GetWorkingDirectory(void);                        // Get current working directory (uses static string)
const char *GetApplicationDirectory(void);                   // Get the directory of the running application (uses static string)
bool ChangeDirectory(const char *dir);                        // Change working directory, return true on success
bool IsPathFile(const char *path);                           // Check if a given path is a file or a directory
FilePathList LoadDirectoryFiles(const char *dirPath);        // Load directory filepaths
FilePathList LoadDirectoryFilesEx(const char *basePath, const char *filter, bool scanSubdirs); // Load directory filepaths with extension filter
void UnloadDirectoryFiles(FilePathList files);              // Unload filepaths
bool IsFileDropped(void);                                    // Check if a file has been dropped into window
FilePathList LoadDroppedFiles(void);                        // Load dropped filepaths
void UnloadDroppedFiles(FilePathList files);                // Unload dropped filepaths
long GetFileModTime(const char *fileName);                  // Get file modification time (last write time)

// Compression/Encoding functionality
unsigned char *CompressData(const unsigned char *data, int dataSize, int *compDataSize); // Compress data (DEFLATE algorithm), memory must be aligned
unsigned char *DecompressData(const unsigned char *compData, int compDataSize, int *dataSize); // Decompress data (DEFLATE algorithm), memory must be aligned
char *EncodeDataBase64(const unsigned char *data, int dataSize, int *outputSize);           // Encode data to Base64 string, memory must be aligned
unsigned char *DecodeDataBase64(const unsigned char *data, int *outputSize);                // Decode Base64 string data, memory must be aligned

// Automation events functionality
AutomationEventList LoadAutomationEventList(const char *fileName); // Load automation events list from file, NULL for empty list
void UnloadAutomationEventList(AutomationEventList *list);        // Unload automation events list from file
bool ExportAutomationEventList(AutomationEventList list, const char *fileName); // Export automation events list as text file
void SetAutomationEventList(AutomationEventList *list);           // Set automation event list to record to
void SetAutomationEventBaseFrame(int frame);                      // Set automation event internal base frame to start recording
void StartAutomationEventRecording(void);                         // Start recording automation events (AutomationEventList)
void StopAutomationEventRecording(void);                          // Stop recording automation events
void PlayAutomationEvent(AutomationEvent event);                // Play a recorded automation event

//-----
// Input Handling Functions (Module: core)
//-----

// Input-related functions: keyboard
bool IsKeyPressed(int key);                                     // Check if a key has been pressed once
bool IsKeyPressedRepeat(int key);                            // Check if a key has been pressed again (Only PLATFORM_DESKTOP)
bool IsKeyDown(int key);                                    // Check if a key is being pressed
bool IsKeyReleased(int key);                                // Check if a key has been released once
bool IsKeyUp(int key);                                     // Check if a key is NOT being pressed
int GetKeyPressed(void);                                    // Get key pressed (keycode), call it multiple times for keys queued, returns 0 when no key is pressed
int GetCharPressed(void);                                  // Get char pressed (unicode), call it multiple times for chars queued, returns 0 when no char is pressed
void SetExitKey(int key);                                  // Set a custom key to exit program (default is ESC)

```

```

// Input-related functions: gamepads
bool IsGamepadAvailable(int gamepad); // Check if a gamepad is available
const char *GetGamepadName(int gamepad); // Get gamepad internal name id
bool IsGamepadButtonPressed(int gamepad, int button); // Check if a gamepad button has been pressed once
bool IsGamepadButtonDown(int gamepad, int button); // Check if a gamepad button is being pressed
bool IsGamepadButtonReleased(int gamepad, int button); // Check if a gamepad button has been released once
bool IsGamepadButtonUp(int gamepad, int button); // Check if a gamepad button is NOT being pressed
int GetGamepadButtonPressed(void); // Get the last gamepad button pressed
int GetGamepadAxisCount(int gamepad); // Get gamepad axis count for a gamepad
float GetGamepadAxisMovement(int gamepad, int axis); // Get axis movement value for a gamepad axis
int SetGamepadMappings(const char *mappings); // Set internal gamepad mappings (SDL_GameControllerDB)

// Input-related functions: mouse
bool IsMouseButtonPressed(int button); // Check if a mouse button has been pressed once
bool IsMouseButtonDown(int button); // Check if a mouse button is being pressed
bool IsMouseButtonReleased(int button); // Check if a mouse button has been released once
bool IsMouseButtonUp(int button); // Check if a mouse button is NOT being pressed
int GetMouseX(void); // Get mouse position X
int GetMouseY(void); // Get mouse position Y
Vector2 GetMousePosition(void); // Get mouse position XY
Vector2 GetMouseDelta(void); // Get mouse delta between frames
void SetMousePosition(int x, int y); // Set mouse position XY
void SetMouseOffset(int offsetX, int offsetY); // Set mouse offset
void SetMouseScale(float scaleX, float scaleY); // Set mouse scaling
float GetMouseWheelMove(void); // Get mouse wheel movement for X or Y, whichever is larger
Vector2 GetMouseWheelMoveV(void); // Get mouse wheel movement for both X and Y
void SetMouseCursor(int cursor); // Set mouse cursor

// Input-related functions: touch
int GetTouchX(void); // Get touch position X for touch point 0 (relative to screen size)
int GetTouchY(void); // Get touch position Y for touch point 0 (relative to screen size)
Vector2 GetTouchPosition(int index); // Get touch position XY for a touch point index (relative to screen size)
int GetTouchPointId(int index); // Get touch point identifier for given index
int GetTouchPointCount(void); // Get number of touch points

//-----
// Gestures and Touch Handling Functions (Module: rgestures)
//-----
void SetGesturesEnabled(unsigned int flags); // Enable a set of gestures using flags
bool IsGestureDetected(unsigned int gesture); // Check if a gesture have been detected
int GetGestureDetected(void); // Get latest detected gesture
float GetGestureHoldDuration(void); // Get gesture hold time in milliseconds
Vector2 GetGestureDragVector(void); // Get gesture drag vector
float GetGestureDragAngle(void); // Get gesture drag angle
Vector2 GetGesturePinchVector(void); // Get gesture pinch delta
float GetGesturePinchAngle(void); // Get gesture pinch angle

//-----
// Camera System Functions (Module: rcamera)
//-----
void UpdateCamera(Camera *camera, int mode); // Update camera position for selected mode
void UpdateCameraPro(Camera *camera, Vector3 movement, Vector3 rotation, float zoom); // Update camera movement/rotation

```

module: rshapes -

```

// NOTE: It can be useful when using basic shapes and one single font,
// defining a font char white rectangle would allow drawing everything in a single draw call
// defining a font char white rectangle would allow drawing everything in a single draw call
void SetShapesTexture(Texture2D texture, Rectangle source); // Set texture and rectangle to be used on shapes drawing

// Basic shapes drawing functions
void DrawPixel(int posX, int posY, Color color); // Draw a pixel
void DrawPixelV(Vector2 position, Color color); // Draw a pixel (Vector version)
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color); // Draw a line
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color); // Draw a line (using gl lines)
void DrawLineEx(Vector2 startPos, Vector2 endPos, float thick, Color color); // Draw a line (using triangles/quads)
void DrawLineStrip(Vector2 *points, int pointCount, Color color); // Draw lines sequence (using gl lines)
void DrawBezier(Vector2 startPos, Vector2 endPos, float thick, Color color); // Draw line segment cubic-bezier in-out
void DrawCircle(int centerX, int centerY, float radius, Color color); // Draw a color-filled circle
void DrawCircleSector(Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color); // Draw a piece of a circle
void DrawCircleSectorLines(Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color); // Draw circle sector
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2); // Draw a gradient-filled circle
void DrawCircleV(Vector2 center, float radius, Color color); // Draw a color-filled circle (Vector version)
void DrawCircleLines(int centerX, int centerY, float radius, Color color); // Draw circle outline
void DrawCircleLinesV(Vector2 center, float radius, Color color); // Draw circle outline (Vector version)
void DrawEllipse(int centerX, int centerY, float radiusH, float radiusV, Color color); // Draw ellipse
void DrawEllipseLines(int centerX, int centerY, float radiusH, float radiusV, Color color); // Draw ellipse outline
void DrawRing(Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color); // Draw ring
void DrawRingLines(Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color); // Draw ring lines
void DrawRectangle(int posX, int posY, int width, int height, Color color); // Draw a color-filled rectangle
void DrawRectangleV(Vector2 position, Vector2 size, Color color); // Draw a color-filled rectangle (Vector version)
void DrawRectangleRec(Rectangle rec, Color color); // Draw a color-filled rectangle
void DrawRectanglePro(Rectangle rec, Vector2 origin, float rotation, Color color); // Draw a color-filled rectangle with perspective
void DrawRectangleGradientV(int posX, int posY, int width, int height, Color color1, Color color2); // Draw a vertical-gradient-filled rectangle
void DrawRectangleGradientH(int posX, int posY, int width, int height, Color color1, Color color2); // Draw a horizontal-gradient-filled rectangle
void DrawRectangleGradientEx(Rectangle rec, Color col1, Color col2, Color col3, Color col4); // Draw a gradient-filled rectangle with extended points
void DrawRectangleLines(int posX, int posY, int width, int height, Color color); // Draw rectangle outline
void DrawRectangleLinesEx(Rectangle rec, float lineThick, Color color); // Draw rectangle outline with extended points
void DrawRectangleRounded(Rectangle rec, float roundness, int segments, Color color); // Draw rectangle with rounded edges
void DrawRectangleRoundedLines(Rectangle rec, float roundness, int segments, float lineThick, Color color); // Draw rectangle with rounded edges
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color); // Draw a color-filled triangle (vertex 1)
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color); // Draw triangle outline (vertex in count)
void DrawTriangleFan(Vector2 *points, int pointCount, Color color); // Draw a triangle fan defined by points
void DrawTriangleStrip(Vector2 *points, int pointCount, Color color); // Draw a triangle strip defined by point
void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color); // Draw a regular polygon (Vector version)
void DrawPolyLines(Vector2 center, int sides, float radius, float rotation, Color color); // Draw a polygon outline of n sides
void DrawPolyLinesEx(Vector2 center, int sides, float radius, float rotation, float lineThick, Color color); // Draw a polygon outline of n sides

// Splines drawing functions
void DrawSplineLinear(Vector2 *points, int pointCount, float thick, Color color); // Draw spline: Linear, minimum 2 points
void DrawSplineBasis(Vector2 *points, int pointCount, float thick, Color color); // Draw spline: B-Spline, minimum 4 points
void DrawSplineCatmullRom(Vector2 *points, int pointCount, float thick, Color color); // Draw spline: Catmull-Rom, minimum 4 points
void DrawSplineBezierQuadratic(Vector2 *points, int pointCount, float thick, Color color); // Draw spline: Quadratic Bezier, minimum 3 points
void DrawSplineBezierCubic(Vector2 *points, int pointCount, float thick, Color color); // Draw spline: Cubic Bezier, minimum 4 points
void DrawSplineSegmentLinear(Vector2 p1, Vector2 p2, float thick, Color color); // Draw spline segment: Linear, 2 points
void DrawSplineSegmentBasis(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color); // Draw spline segment: B-Spline, 4 points
void DrawSplineSegmentCatmullRom(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color); // Draw spline segment: Catmull-Rom, 4 points
void DrawSplineSegmentBezierQuadratic(Vector2 p1, Vector2 p2, Vector2 p3, float thick, Color color); // Draw spline segment: Quadratic Bezier, 3 points
void DrawSplineSegmentBezierCubic(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color); // Draw spline segment: Cubic Bezier, 4 points

```

```

// Spline segment point evaluation functions, for a given t [0.0f .. 1.0f]
Vector2 GetSplinePointLinear(Vector2 startPos, Vector2 endPos, float t);
Vector2 GetSplinePointBasis(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float t);
Vector2 GetSplinePointCatmullRom(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float t);
Vector2 GetSplinePointBezierQuad(Vector2 p1, Vector2 c2, Vector2 p3, float t);
Vector2 GetSplinePointBezierCubic(Vector2 p1, Vector2 c2, Vector2 c3, Vector2 p4, float t);

// Basic shapes collision detection functions
bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2);
bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2);
bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec);
bool CheckCollisionPointRec(Vector2 point, Rectangle rec);
bool CheckCollisionPointCircle(Vector2 point, Vector2 center, float radius);
bool CheckCollisionPointTriangle(Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3);
bool CheckCollisionPointPoly(Vector2 point, Vector2 *points, int pointCount);
bool CheckCollisionLines(Vector2 startPos1, Vector2 endPos1, Vector2 startPos2, Vector2 endPos2, Vector2 *collisionPoint); // Check the collision
bool CheckCollisionPointLine(Vector2 point, Vector2 p1, Vector2 p2, int threshold);
Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2);

// Get (evaluate) spline point: Linear
// Get (evaluate) spline point: B-Spline
// Get (evaluate) spline point: Catmull-Rom
// Get (evaluate) spline point: Quadratic
// Get (evaluate) spline point: Cubic Bezier

```

module: rtextures →

```

// Image loading functions
// NOTE: These functions do not require GPU access
Image LoadImage(const char *fileName);
Image LoadImageRaw(const char *fileName, int width, int height, int format, int headerSize);
Image LoadImageSvg(const char *fileNameOrString, int width, int height);
Image LoadImageAnim(const char *fileName, int *frames);
Image LoadImageFromMemory(const char * fileType, const unsigned char *fileData, int dataSize);
Image LoadImageFromTexture(Texture2D texture);
Image LoadImageFromScreen(void);
bool IsImageReady(Image image);
void UnloadImage(Image image);
bool ExportImage(Image image, const char *fileName);
unsigned char *ExportImageToMemory(Image image, const char *fileType, int *fileSize);
bool ExportImageAsCode(Image image, const char *fileName);

// Image generation functions
Image GenImageColor(int width, int height, Color color);
Image GenImageGradientLinear(int width, int height, int direction, Color start, Color end);
Image GenImageGradientRadial(int width, int height, float density, Color inner, Color outer);
Image GenImageGradientSquare(int width, int height, float density, Color inner, Color outer);
Image GenImageChecked(int width, int height, int checksX, int checksY, Color coll, Color col2);
Image GenImageWhiteNoise(int width, int height, float factor);
Image GenImagePerlinNoise(int width, int height, int offsetX, int offsetY, float scale);
Image GenImageCellular(int width, int height, int tileSize);
Image GenImageText(int width, int height, const char *text);

// Image manipulation functions
Image ImageCopy(Image image);
Image ImageFromImage(Image image, Rectangle rec);
Image ImageText(const char *text, int fontSize, Color color);
Image ImageTextEx(Font font, const char *text, float fontSize, float spacing, Color tint);
void ImageFormat(Image *image, int newFormat);
void ImageToPOT(Image *image, Color fill);
void ImageCrop(Image *image, Rectangle crop);
void ImageAlphaCrop(Image *image, float threshold);
void ImageAlphaClear(Image *image, Color color, float threshold);
void ImageAlphaMask(Image *image, Image alphaMask);
void ImageAlphaPremultiply(Image *image);
void ImageBlurGaussian(Image *image, int blurSize);
void ImageResize(Image *image, int newWidth, int newHeight);
void ImageResizeNN(Image *image, int newWidth, int newHeight);
void ImageResizeCanvas(Image *image, int newWidth, int newHeight, int offsetX, int offsetY, Color fill);
void ImageMipmaps(Image *image);
void ImageDither(Image *image, int rBpp, int gBpp, int bBpp, int aBpp);
void ImageFlipVertical(Image *image);
void ImageFlipHorizontal(Image *image);
void ImageRotate(Image *image, int degrees);
void ImageRotateCW(Image *image);
void ImageRotateCCW(Image *image);
void ImageColorTint(Image *image, Color color);
void ImageColorInvert(Image *image);
void ImageColorGrayscale(Image *image);
void ImageColorContrast(Image *image, float contrast);
void ImageColorBrightness(Image *image, int brightness);
void ImageColorReplace(Image *image, Color color, Color replace);
Color *LoadImageColors(Image image);
Color *LoadImagePalette(Image image, int maxPaletteSize, int *colorCount);
void UnloadImageColors(Color *colors);
void UnloadImagePalette(Color *colors);
Rectangle GetImageAlphaBorder(Image image, float threshold);
Color GetImageColor(Image image, int x, int y);

// Image drawing functions
// NOTE: Image software-rendering functions (CPU)
void ImageClearBackground(Image *dst, Color color);
void ImageDrawPixel(Image *dst, int posX, int posY, Color color);
void ImageDrawPixelV(Image *dst, Vector2 position, Color color);
void ImageDrawLine(Image *dst, int startPosX, int startPosY, int endPosX, int endPosY, Color color);
void ImageDrawLineV(Image *dst, Vector2 start, Vector2 end, Color color);
void ImageDrawCircle(Image *dst, int centerX, int centerY, int radius, Color color);
void ImageDrawCircleV(Image *dst, Vector2 center, int radius, Color color);
void ImageDrawCircleLines(Image *dst, int centerX, int centerY, int radius, Color color);
void ImageDrawCircleLinesV(Image *dst, Vector2 center, int radius, Color color);
void ImageDrawRectangle(Image *dst, int posX, int posY, int width, int height, Color color);
void ImageDrawRectangleV(Image *dst, Vector2 position, Vector2 size, Color color);
void ImageDrawRectangleRec(Image *dst, Rectangle rec, Color color);
void ImageDrawRectangleLines(Image *dst, Rectangle rec, int thick, Color color);
void ImageDraw(Image *dst, Image src, Rectangle srcRec, Rectangle dstRec, Color tint);
void ImageDrawText(Image *dst, const char *text, int posX, int posY, int fontSize, Color color); // Draw text (using default font) within
void ImageDrawTextEx(Image *dst, Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint); // Draw text (using

// Texture loading functions
// NOTE: These functions require GPU access
Texture2D LoadTexture(const char *fileName);
Texture2D LoadTextureFromImage(Image image);
TextureCubemap LoadTextureCubemap(Image image, int layout);

// Load texture from file into GPU memory
// Load texture from image data
// Load cubemap from image, multiple images

```

```

RenderTexture2D LoadRenderTexture(int width, int height);
bool IsTextureReady(Texture2D texture);
void UnloadTexture(Texture2D texture);
bool IsRenderTextureReady(RenderTexture2D target);
void UnloadRenderTexture(RenderTexture2D target);
void UpdateTexture(Texture2D texture, const void *pixels);
void UpdateTextureRec(Texture2D texture, Rectangle rec, const void *pixels);

// Texture configuration functions
void GenTextureMipmaps(Texture2D *texture);
void SetTextureFilter(Texture2D texture, int filter);
void SetTextureWrap(Texture2D texture, int wrap);

// Texture drawing functions
void DrawTexture(Texture2D texture, int posX, int posY, Color tint);
void DrawTextureV(Texture2D texture, Vector2 position, Color tint);
void DrawTextureEx(Texture2D texture, Vector2 position, float rotation, float scale, Color tint);
void DrawTextureRec(Texture2D texture, Rectangle source, Vector2 position, Color tint);
void DrawTexturePro(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, Color tint);
void DrawTextureNPatch(Texture2D texture, NPatchInfo npatchInfo, Rectangle dest, Vector2 origin, float rotation, Color tint);

// Color/pixel related functions
Color Fade(Color color, float alpha);
int ColorToInt(Color color);
Vector4 ColorNormalize(Color color);
Color ColorFromNormalized(Vector4 normalized);
Vector3 ColorToHSV(Color color);
Color ColorFromHSV(float hue, float saturation, float value);
Color ColorTint(Color color, Color tint);
Color ColorBrightness(Color color, float factor);
Color ColorContrast(Color color, float contrast);
Color ColorAlpha(Color color, float alpha);
Color ColorAlphaBlend(Color dst, Color src, Color tint);
Color GetColor(unsigned int hexValue);
Color GetPixelColor(void *srcPtr, int format);
void SetPixelColor(void *dstPtr, Color color, int format);
int GetPixelDataSize(int width, int height, int format);

```

```

// Load texture for rendering (framebuffer)
// Check if a texture is ready
// Unload texture from GPU memory (VRAM)
// Check if a render texture is ready
// Unload render texture from GPU memory
// Update GPU texture with new data
// Update GPU texture rectangle with new

// Generate GPU mipmaps for a texture
// Set texture scaling filter mode
// Set texture wrapping mode

// Draw a Texture2D
// Draw a Texture2D with position defined
// Draw a Texture2D with extended parameters
// Draw a part of a texture defined by a rectangle
// Draw a part of a texture
// Draw a part of a texture with rotation
// Draws a texture with rotation and tint

// Get color with alpha applied, alpha goes from 0.0f to 1.0f
// Get hexadecimal value for a Color
// Get Color normalized as float [0..1]
// Get Color from normalized values [0..1]
// Get HSV values for a Color, hue [0..360], saturation/value [0..1]
// Get a Color from HSV values, hue [0..360], saturation/value [0..1]
// Get color multiplied with another color
// Get color with brightness correction, brightness factor goes from -1.0f to 1.0f
// Get color with contrast correction, contrast values between -1.0f and 1.0f
// Get color with alpha applied, alpha goes from 0.0f to 1.0f
// Get src alpha-blended into dst color with tint
// Get Color structure from hexadecimal value
// Get Color from a source pixel pointer of certain format
// Set color formatted into destination pixel pointer
// Get pixel data size in bytes for certain format

```

module: rtext -

```

// Font loading/unloading functions
Font GetFontDefault(void); // Get the default Font
Font LoadFont(const char *fileName); // Load font from file into GPU memory (VRAM)
Font LoadFontEx(const char *fileName, int fontSize, int *codepoints, int codepointCount); // Load font from file with extended parameters, // Load font from Image (XNA style)
Font LoadFontFromImage(Image image, Color key, int firstChar); // Load font from Image (XNA style)
Font LoadFontFromMemory(const char *fileType, const unsigned char *fileData, int dataSize, int fontSize, int *codepoints, int codepointCount); // Check if a font is ready
bool IsFontReady(Font font); // Check if a font is ready
GlyphInfo *LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *codepoints, int codepointCount, int type); // Load font data
Image GenImageFontAtlas(const GlyphInfo *glyphs, Rectangle **glyphRecs, int glyphCount, int fontSize, int padding, int packMethod); // Generate font atlas
void UnloadFontData(GlyphInfo *glyphs, int glyphCount); // Unload font chars info data (RAM)
void UnloadFont(Font font); // Unload font from GPU memory (VRAM)
bool ExportFontAsCode(Font font, const char *fileName); // Export font as code file, returns true on success

// Text drawing functions
void DrawFPS(int posX, int posY); // Draw current FPS
void DrawText(const char *text, int posX, int posY, int fontSize, Color color); // Draw text (using default font)
void DrawTextEx(Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint); // Draw text using font and additional parameters
void DrawTextPro(Font font, const char *text, Vector2 position, Vector2 origin, float rotation, float fontSize, float spacing, Color tint);
void DrawTextCodepoint(Font font, int codepoint, Vector2 position, float fontSize, Color tint); // Draw one character (codepoint)
void DrawTextCodepoints(Font font, const int *codepoints, int codepointCount, Vector2 position, float fontSize, float spacing, Color tint);

// Text font info functions
void SetTextLineSpacing(int spacing); // Set vertical line spacing when drawing with line-based font
int MeasureText(const char *text, int fontSize); // Measure string width for default font
Vector2 MeasureTextEx(Font font, const char *text, float fontSize, float spacing); // Measure string size for Font
int GetGlyphIndex(Font font, int codepoint); // Get glyph index position in font for a codepoint (unicode code)
GlyphInfo GetGlyphInfo(Font font, int codepoint); // Get glyph font info data for a codepoint (unicode code)
Rectangle GetGlyphAtlasRec(Font font, int codepoint); // Get glyph rectangle in font atlas for a codepoint (unicode code)

// Text codepoints management functions (unicode characters)
char *LoadUTF8(const int *codepoints, int length); // Load UTF-8 text encoded from codepoints array
void UnloadUTF8(char *text); // Unload UTF-8 text encoded from codepoints array
int *LoadCodepoints(const char *text, int *count); // Load all codepoints from a UTF-8 text string, codepoints count returned
void UnloadCodepoints(int *codepoints); // Unload codepoints data from memory
int GetCodepointCount(const char *text); // Get total number of codepoints in a UTF-8 encoded string
int GetCodepoint(const char *text, int *codepointSize); // Get next codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
int GetCodepointNext(const char *text, int *codepointSize); // Get next codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
int GetCodepointPrevious(const char *text, int *codepointSize); // Get previous codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
const char *CodepointToUTF8(int codepoint, int *utf8Size); // Encode one codepoint into UTF-8 byte array (array length returned as parameter)

// Text strings management functions (no UTF-8 strings, only byte chars)
// NOTE: Some strings allocate memory internally for returned strings, just be careful!
int TextCopy(char *dst, const char *src); // Copy one string to another, returns bytes copied
bool TextIsEqual(const char *text1, const char *text2); // Check if two text strings are equal
unsigned int TextLength(const char *text); // Get text length, checks for '\0' ending
const char *TextFormat(const char *text, ...); // Text formatting with variables (sprintf() style)
const char *TextSubtext(const char *text, int position, int length); // Get a piece of a text string
char *TextReplace(char *text, const char *replace, const char *by); // Replace text string (WARNING: memory must be freed)
char *TextInsert(const char *text, const char *insert, int position); // Insert text in a position (WARNING: memory must be freed)
const char *TextJoin(const char **textList, int count, const char *delimiter); // Join text strings with delimiter
const char **TextSplit(const char *text, char delimiter, int *count); // Split text into multiple strings
void TextAppend(char *text, const char *append, int *position); // Append text at specific position and move cursor!
int TextFindIndex(const char *text, const char *find); // Find first text occurrence within a string
const char *TextToUpper(const char *text); // Get upper case version of provided string
const char *TextToLower(const char *text); // Get lower case version of provided string
const char *TextToPascal(const char *text); // Get Pascal case notation version of provided string
int TextToInteger(const char *text); // Get integer value from text (negative values not supported)

```

module: rmodels -

```

// Basic geometric 3D shapes drawing functions
void DrawLine3D(Vector3 startPos, Vector3 endPos, Color color); // Draw a line in 3D world space
void DrawPoint3D(Vector3 position, Color color); // Draw a point in 3D space, actually a small sphere
void DrawCircle3D(Vector3 center, float radius, Vector3 rotationAxis, float rotationAngle, Color color); // Draw a circle in 3D world space

```

```

void DrawTriangle3D(Vector3 v1, Vector3 v2, Vector3 v3, Color color);
void DrawTriangleStrip3D(Vector3 *points, int pointCount, Color color);
void DrawCube(Vector3 position, float width, float height, float length, Color color);
void DrawCubeV(Vector3 position, Vector3 size, Color color);
void DrawCubeWires(Vector3 position, float width, float height, float length, Color color);
void DrawCubeWiresV(Vector3 position, Vector3 size, Color color);
void DrawSphere(Vector3 centerPos, float radius, Color color);
void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color color);
void DrawSphereWires(Vector3 centerPos, float radius, int rings, int slices, Color color);
void DrawCylinder(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone
void DrawCylinderEx(Vector3 startPos, Vector3 endPos, float startRadius, float endRadius, int sides, Color color); // Draw a cylinder with b
void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/con
void DrawCylinderWiresEx(Vector3 startPos, Vector3 endPos, float startRadius, float endRadius, int sides, Color color); // Draw a cylinder w
void DrawCapsule(Vector3 startPos, Vector3 endPos, float radius, int slices, int rings, Color color); // Draw a capsule with the center of i
void DrawCapsuleWires(Vector3 startPos, Vector3 endPos, float radius, int slices, int rings, Color color); // Draw capsule wireframe with th
void DrawPlane(Vector3 centerPos, Vector2 size, Color color); // Draw a plane XZ
void DrawRay(Ray ray, Color color); // Draw a ray line
void DrawGrid(int slices, float spacing); // Draw a grid (centered at (0, 0, 0))

//-----
// Model 3d Loading and Drawing Functions (Module: models)
//-----
```

```

// Model management functions
Model LoadModel(const char *fileName);
Model LoadModelFromMesh(Mesh mesh); // Load model from files (meshes and materials)
Model LoadModelFromGeneratedMesh(Mesh mesh); // Load model from generated mesh (default material)
bool IsModelReady(Model model); // Check if a model is ready
void UnloadModel(Model model); // Unload model (including meshes) from memory (RAM and GPU)
BoundingBox GetModelBoundingBox(Model model); // Compute model bounding box limits (considers all meshes)

// Model drawing functions
void DrawModel(Model model, Vector3 position, float scale, Color tint); // Draw a model (with texture if set)
void DrawModelEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model with
void DrawModelWires(Model model, Vector3 position, float scale, Color tint); // Draw a model wires (with texture if set)
void DrawModelWiresEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model
void DrawBoundingBox(BoundingBox box, Color color); // Draw bounding box (wires)
void DrawBillboard(Camera camera, Texture2D texture, Vector3 position, float size, Color tint); // Draw a billboard texture
void DrawBillboardRec(Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector2 size, Color tint); // Draw a billboard t
void DrawBillboardPro(Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector2 up, Vector2 size, Vector2 origin, float

// Mesh management functions
void UploadMesh(Mesh *mesh, bool dynamic); // Upload mesh vertex data in GPU and provide VAO/VBO
void UpdateMeshBuffer(Mesh mesh, int index, const void *data, int dataSize, int offset); // Update mesh vertex data in GPU for a specific bu
void UnloadMesh(Mesh mesh); // Unload mesh data from CPU and GPU
void DrawMesh(Mesh mesh, Material material, Matrix transform); // Draw a 3d mesh with material and transform
void DrawMeshInstanced(Mesh mesh, Material material, const Matrix *transforms, int instances); // Draw multiple mesh instances with material
bool ExportMesh(Mesh mesh, const char *fileName); // Export mesh data to file, returns true on success
BoundingBox GetMeshBoundingBox(Mesh mesh); // Compute mesh bounding box limits
void GenMeshTangents(Mesh *mesh); // Compute mesh tangents

// Mesh generation functions
Mesh GenMeshPoly(int sides, float radius); // Generate polygonal mesh
Mesh GenMeshPlane(float width, float length, int resX, int resZ); // Generate plane mesh (with subdivisions)
Mesh GenMeshCube(float width, float height, float length); // Generate cuboid mesh
Mesh GenMeshSphere(float radius, int rings, int slices); // Generate sphere mesh (standard sphere)
Mesh GenMeshHemiSphere(float radius, int rings, int slices); // Generate half-sphere mesh (no bottom cap)
Mesh GenMeshCylinder(float radius, float height, int slices); // Generate cylinder mesh
Mesh GenMeshCone(float radius, float height, int slices); // Generate cone/pyramid mesh
Mesh GenMeshTorus(float radius, float size, int radSeg, int sides); // Generate torus mesh
Mesh GenMeshKnot(float radius, float size, int radSeg, int sides); // Generate trefoil knot mesh
Mesh GenMeshHeightmap(Image heightmap, Vector3 size); // Generate heightmap mesh from image data
Mesh GenMeshCubemap(Image cubemap, Vector3 cubeSize); // Generate cubes-based map mesh from image data

// Material loading/unloading functions
Material *LoadMaterials(const char *fileName, int *materialCount); // Load materials from model file
Material LoadMaterialDefault(void); // Load default material (Supports: DIFFUSE, SPECULAR, ETC)
bool IsMaterialReady(Material material); // Check if a material is ready
void UnloadMaterial(Material material); // Unload material from GPU memory (VRAM)
void SetMaterialTexture(Material *material, int mapType, Texture2D texture); // Set texture for a material map type (MATERIAL_MAP_1)
void SetModelMeshMaterial(Model *model, int meshId, int materialId); // Set material for a mesh

// Model animations loading/unloading functions
ModelAnimation *LoadModelAnimations(const char *fileName, int *animCount); // Load model animations from file
void UpdateModelAnimation(Model model, ModelAnimation anim, int frame); // Update model animation pose
void UnloadModelAnimation(ModelAnimation anim); // Unload animation data
void UnloadModelAnimations(ModelAnimation *animations, int animCount); // Unload animation array data
bool IsModelAnimationValid(Model model, ModelAnimation anim); // Check model animation skeleton match

// Collision detection functions
bool CheckCollisionSpheres(Vector3 center1, float radius1, Vector3 center2, float radius2); // Check collision between two spheres
bool CheckCollisionBoxes(BoundingBox box1, BoundingBox box2); // Check collision between two bounding boxes
bool CheckCollisionBoxSphere(BoundingBox box, Vector3 center, float radius); // Check collision between box and sphere
RayCollision GetRayCollisionSphere(Ray ray, Vector3 center, float radius); // Get collision info between ray and sphere
RayCollision GetRayCollisionBox(Ray ray, BoundingBox box); // Get collision info between ray and box
RayCollision GetRayCollisionMesh(Ray ray, Mesh mesh, Matrix transform); // Get collision info between ray and mesh
RayCollision GetRayCollisionTriangle(Ray ray, Vector3 p1, Vector3 p2, Vector3 p3); // Get collision info between ray and triangle
RayCollision GetRayCollisionQuad(Ray ray, Vector3 p1, Vector3 p2, Vector3 p3, Vector3 p4); // Get collision info between ray and quad
```

module: raudio -

```

// Audio device management functions
void InitAudioDevice(void); // Initialize audio device and context
void CloseAudioDevice(void); // Close the audio device and context
bool IsAudioDeviceReady(void); // Check if audio device has been initialized successfully
void SetMasterVolume(float volume); // Set master volume (listener)
float GetMasterVolume(void); // Get master volume (listener)

// Wave/Sound loading/unloading functions
Wave LoadWave(const char *fileName); // Load wave data from file
Wave LoadWaveFromMemory(const char *fileType, const unsigned char *fileData, int dataSize); // Load wave from memory buffer, fileType refers to the file type
bool IsWaveReady(Wave wave); // Checks if wave data is ready
Sound LoadSound(const char *fileName); // Load sound from file
Sound LoadSoundFromWave(Wave wave); // Load sound from wave data
Sound LoadSoundAlias(Sound source); // Create a new sound that shares the same sample data as the source sound, but with a different ID
bool IsSoundReady(Sound sound); // Checks if a sound is ready
void UpdateSound(Sound sound, const void *data, int sampleCount); // Update sound buffer with new data
void UnloadWave(Wave wave); // Unload wave data
void UnloadSound(Sound sound); // Unload sound
```

```

void UnloadSoundAlias(Sound alias);
bool ExportWave(Wave wave, const char *fileName);
bool ExportWaveAsCode(Wave wave, const char *fileName);

// Wave/Sound management functions
void PlaySound(Sound sound);
void StopSound(Sound sound);
void PauseSound(Sound sound);
void ResumeSound(Sound sound);
bool IsSoundPlaying(Sound sound);
void SetSoundVolume(Sound sound, float volume);
void SetSoundPitch(Sound sound, float pitch);
void SetSoundPan(Sound sound, float pan);
Wave WaveCopy(Wave wave);
void WaveCrop(Wave *wave, int initSample, int finalSample);
void WaveFormat(Wave *wave, int sampleRate, int sampleSize, int channels);
float *LoadWaveSamples(Wave wave);
void UnloadWaveSamples(float *samples);

// Music management functions
Music LoadMusicStream(const char *fileName);
Music LoadMusicStreamFromMemory(const char *fileType, const unsigned char *data, int dataSize);
bool IsMusicReady(Music music);
void UnloadMusicStream(Music music);
void PlayMusicStream(Music music);
bool IsMusicStreamPlaying(Music music);
void UpdateMusicStream(Music music);
void StopMusicStream(Music music);
void PauseMusicStream(Music music);
void ResumeMusicStream(Music music);
void SeekMusicStream(Music music, float position);
void SetMusicVolume(Music music, float volume);
void SetMusicPitch(Music music, float pitch);
void SetMusicPan(Music music, float pan);
float GetMusicTimeLength(Music music);
float GetMusicTimePlayed(Music music);

// AudioStream management functions
AudioStream LoadAudioStream(unsigned int sampleRate, unsigned int sampleSize, unsigned int channels);
bool IsAudioStreamReady(AudioStream stream);
void UnloadAudioStream(AudioStream stream);
void UpdateAudioStream(AudioStream stream, const void *data, int frameCount);
bool IsAudioStreamProcessed(AudioStream stream);
void PlayAudioStream(AudioStream stream);
void PauseAudioStream(AudioStream stream);
void ResumeAudioStream(AudioStream stream);
bool IsAudioStreamPlaying(AudioStream stream);
void StopAudioStream(AudioStream stream);
void SetAudioStreamVolume(AudioStream stream, float volume);
void SetAudioStreamPitch(AudioStream stream, float pitch);
void SetAudioStreamPan(AudioStream stream, float pan);
void SetAudioStreamBufferSizeDefault(int size);
void SetAudioStreamCallback(AudioStream stream, AudioCallback callback);

void AttachAudioStreamProcessor(AudioStream stream, AudioCallback processor);
void DetachAudioStreamProcessor(AudioStream stream, AudioCallback processor);

void AttachAudioMixedProcessor(AudioCallback processor);
void DetachAudioMixedProcessor(AudioCallback processor);

```

structs

```

struct Vector2;           // Vector2, 2 components
struct Vector3;           // Vector3, 3 components
struct Vector4;           // Vector4, 4 components
struct Matrix;            // Matrix, 4x4 components, column major, OpenGL style, right handed
struct Color;              // Color, 4 components, R8G8B8A8 (32bit)
struct Rectangle;          // Rectangle, 4 components

struct Image;              // Image, pixel data stored in CPU memory (RAM)
struct Texture;            // Texture, tex data stored in GPU memory (VRAM)
struct RenderTexture;      // RenderTexture, fbo for texture rendering
struct NPatchInfo;         // NPatchInfo, n-patch layout info
struct GlyphInfo;           // GlyphInfo, font characters glyphs info
struct Font;                // Font, font texture and GlyphInfo array data

struct Camera3D;           // Camera, defines position/orientation in 3d space

struct Camera2D;           // Camera2D, defines position/orientation in 2d space
struct Mesh;                // Mesh, vertex data and vao/vbo
struct Shader;              // Shader
struct MaterialMap;         // MaterialMap
struct Material;            // Material, includes shader and maps
struct Transform;           // Transform, vertex transformation data
struct BoneInfo;             // Bone, skeletal animation bone
struct Model;                // Model, meshes, materials and animation data
struct ModelAnimation;       // ModelAnimation
struct Ray;                  // Ray, ray for raycasting
struct RayCollision;         // RayCollision, ray hit information
struct BoundingBox;          // BoundingBox

struct Wave;                  // Wave, audio wave data
struct AudioStream;          // AudioStream, custom audio stream
struct Sound;                // Sound
struct Music;                // Music, audio stream, anything longer than ~10 seconds should be streamed

struct VrDeviceInfo;          // VrDeviceInfo, Head-Mounted-Display device parameters
struct VrStereoConfig;        // VrStereoConfig, VR stereo rendering configuration for simulator

struct FilePathList;          // File path list

struct AutomationEvent;       // Automation event
struct AutomationEventList;    // Automation event list

```

colors

```

// Custom raylib color palette
#define LIGHTGRAY (Color)
#define GRAY (Color)
#define DARKGRAY (Color)
#define YELLOW (Color)
#define GOLD (Color)
#define ORANGE (Color)
#define PINK (Color)
#define RED (Color)
#define MAROON (Color)
#define GREEN (Color)
#define LIME (Color)
#define DARKGREEN (Color)
#define SKYBLUE (Color)
#define BLUE (Color)
#define DARKBLUE (Color)
#define PURPLE (Color)
#define VIOLET (Color)
#define DARKPURPLE (Color)
#define BEIGE (Color)
#define BROWN (Color)
#define DARKBROWN (Color)

#define WHITE (Color)
#define BLACK (Color)
#define BLANK (Color)
#define MAGENTA (Color)
#define RAYWHITE (Color)

```

- [raymath cheatsheet](#)

raylib quick reference card - Copyright (c) 2013-2024 Ramon Santamaría (@raysan5)