raylib

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

[raylib Discord server][github.com/raysan5/raylib]

v3.0 quick reference card [download as PDF]

module: core

```
// Window-related functions
void InitWindow(int width, int height, const char *title);              // Initialize window and OpenGL context
bool WindowShouldClose(void);                                           // Check if KEY_ESCAPE pressed or Close icon pressed
void CloseWindow(void);                                                 // Close window and unload OpenGL context
bool IsWindowReady(void);                                               // Check if window has been initialized successfully
bool IsWindowMinimized(void);                                           // Check if window has been minimized (or lost focus)
bool IsWindowResized(void);                                             // Check if window has been resized
bool IsWindowHidden(void);                                              // Check if window is currently hidden
void ToggleFullscreen(void);                                            // Toggle fullscreen mode (only PLATFORM_DESKTOP)
void UnhideWindow(void);                                                // Show the window
void HideWindow(void);                                                  // Hide the window
void SetWindowIcon(Image image);                                        // Set icon for window (only PLATFORM_DESKTOP)
void SetWindowTitle(const char *title);                                 // Set title for window (only PLATFORM_DESKTOP)
void SetWindowPosition(int x, int y);                                   // Set window position on screen (only PLATFORM_DESKTOP)
void SetWindowMonitor(int monitor);                                     // Set monitor for the current window (fullscreen mode)
void SetWindowMinSize(int width, int height);                           // Set window minimum dimensions (for FLAG_WINDOW_RESIZABLE)
void SetWindowSize(int width, int height);                              // Set window dimensions
void *GetWindowHandle(void);                                            // Get native window handle
int GetScreenWidth(void);                                               // Get current screen width
int GetScreenHeight(void);                                              // Get current screen height
int GetMonitorCount(void);                                              // Get number of connected monitors
int GetMonitorWidth(int monitor);                                       // Get primary monitor width
int GetMonitorHeight(int monitor);                                      // Get primary monitor height
int GetMonitorPhysicalWidth(int monitor);                               // Get primary monitor physical width in millimetres
int GetMonitorPhysicalHeight(int monitor);                              // Get primary monitor physical height in millimetres
Vector2 GetWindowPosition(void);                                        // Get window position XY on monitor
const char *GetMonitorName(int monitor);                                // Get the human-readable, UTF-8 encoded name of the primary monitor
const char *GetClipboardText(void);                                     // Get clipboard text content
void SetClipboardText(const char *text);                                // Set clipboard text content

// 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)

// 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);                                      // Initialize 2D mode with custom camera (2D)
void EndMode2D(void);                                                   // Ends 2D mode with custom camera
void BeginMode3D(Camera3D camera);                                      // Initializes 3D mode with custom camera (3D)
void EndMode3D(void);                                                   // Ends 3D mode and returns to default 2D orthographic mode
void BeginTextureMode(RenderTexture2D target);                          // Initializes render texture for drawing
void EndTextureMode(void);                                              // Ends drawing to render texture
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

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

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

// Color-related functions
int ColorToInt(Color color);                                            // Returns hexadecimal value for a Color
Vector4 ColorNormalize(Color color);                                    // Returns color normalized as float [0..1]
Color ColorFromNormalized(Vector4 normalized);                          // Returns color from normalized values [0..1]
Vector3 ColorToHSV(Color color);                                        // Returns HSV values for a Color
Color ColorFromHSV(Vector3 hsv);                                        // Returns a Color from HSV values
Color GetColor(int hexValue);                                           // Returns a Color struct from hexadecimal value
Color Fade(Color color, float alpha);                                   // Color fade-in or fade-out, alpha goes from 0.0f to 1.0f

// Misc. functions
void SetConfigFlags(unsigned int flags);                                // Setup window configuration flags (view FLAGS)
void SetTraceLogLevel(int logType);                                     // Set the current threshold (minimum) log level
void SetTraceLogExit(int logType);                                      // Set the exit threshold (minimum) log level
void SetTraceLogCallback(TraceLogCallback callback);                    // Set a trace log callback to enable custom logging
void TraceLog(int logType, const char *text, ...);                      // Show trace log messages (LOG_DEBUG, LOG_INFO, LOG_WARNING, LOG_ERROR)
void TakeScreenshot(const char *fileName);                              // Takes a screenshot of current screen (saved a .png)
int GetRandomValue(int min, int max);                                   // Returns a random value between min and max (both included)

// Files management functions
bool FileExists(const char *fileName);                                  // Check if file exists
bool IsFileExtension(const char *fileName, const char *ext);            // Check file extension
bool DirectoryExists(const char *dirPath);                              // Check if a directory path exists
const char *GetExtension(const char *fileName);                         // Get pointer to extension for a filename string
const char *GetFileName(const char *filePath);                          // 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)
char **GetDirectoryFiles(const char *dirPath, int *count);              // Get filenames in a directory path (memory should be freed)
void ClearDirectoryFiles(void);                                         // Clear directory files paths buffers (free memory)
bool ChangeDirectory(const char *dir);                                  // Change working directory, returns true if success
bool IsFileDropped(void);                                               // Check if a file has been dropped into window
char **GetDroppedFiles(int *count);                                     // Get dropped files names (memory should be freed)
void ClearDroppedFiles(void);                                           // Clear dropped files paths buffer (free memory)
long GetFileModTime(const char *fileName);                              // Get file modification time (last write time)

unsigned char *CompressData(unsigned char *data, int dataLength, int *compDataLength);        // Compress data (DEFLATE algorythm)
unsigned char *DecompressData(unsigned char *compData, int compDataLength, int *dataLength);  // Decompress data (DEFLATE algorythm)

// Persistent storage management
void StorageSaveValue(int position, int value);                         // Save integer value to storage file (to defined position)
int StorageLoadValue(int position);                                     // Load integer value from storage file (from defined position)

void OpenURL(const char *url);                                          // Open URL with default system browser (if available)

//------------------------------------------------------------------------------------
// Input Handling Functions
//------------------------------------------------------------------------------------

// Input-related functions: keyb
bool IsKeyPressed(int key);                                             // Detect if a key has been pressed once
bool IsKeyDown(int key);                                                // Detect if a key is being pressed
bool IsKeyReleased(int key);                                            // Detect if a key has been released once
bool IsKeyUp(int key);                                                  // Detect if a key is NOT being pressed
int GetKeyPressed(void);                                                // Get latest key pressed
void SetExitKey(int key);                                               // Set a custom key to exit program (default is ESC)

// Input-related functions: gamepads
bool IsGamepadAvailable(int gamepad);                                   // Detect if a gamepad is available
bool IsGamepadName(int gamepad, const char *name);                      // Check gamepad name (if available)
const char *GetGamepadName(int gamepad);                                // Return gamepad internal name id
bool IsGamepadButtonPressed(int gamepad, int button);                   // Detect if a gamepad button has been pressed once
bool IsGamepadButtonDown(int gamepad, int button);                      // Detect if a gamepad button is being pressed
bool IsGamepadButtonReleased(int gamepad, int button);                  // Detect if a gamepad button has been released once
bool IsGamepadButtonUp(int gamepad, int button);                        // Detect if a gamepad button is NOT being pressed
int GetGamepadButtonPressed(void);                                      // Get the last gamepad button pressed
```

```
int GetGamepadAxisCount(int gamepad);                                   // Return gamepad axis count for a gamepad
float GetGamepadAxisMovement(int gamepad, int axis);                    // Return axis movement value for a gamepad axis

// Input-related functions: mouse
bool IsMouseButtonPressed(int button);                                  // Detect if a mouse button has been pressed once
bool IsMouseButtonDown(int button);                                     // Detect if a mouse button is being pressed
bool IsMouseButtonReleased(int button);                                 // Detect if a mouse button has been released once
bool IsMouseButtonUp(int button);                                       // Detect if a mouse button is NOT being pressed
int GetMouseX(void);                                                    // Returns mouse position X
int GetMouseY(void);                                                    // Returns mouse position Y
Vector2 GetMousePosition(void);                                         // Returns mouse position XY
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
int GetMouseWheelMove(void);                                            // Returns mouse wheel movement Y

// Input-related functions: touch
int GetTouchX(void);                                                    // Returns touch position X for touch point 0 (relative to screen size)
int GetTouchY(void);                                                    // Returns touch position Y for touch point 0 (relative to screen size)
Vector2 GetTouchPosition(int index);                                    // Returns touch position XY for a touch point index (relative to screen size)

//------------------------------------------------------------------------------------
// Gestures and Touch Handling Functions (Module: gestures)
//------------------------------------------------------------------------------------
void SetGesturesEnabled(unsigned int gestureFlags);                     // Enable a set of gestures using flags
bool IsGestureDetected(int gesture);                                    // Check if a gesture have been detected
int GetGestureDetected(void);                                           // Get latest detected gesture
int GetTouchPointsCount(void);                                          // Get touch points count
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: camera)
//------------------------------------------------------------------------------------
void SetCameraMode(Camera camera, int mode);                            // Set camera mode (multiple camera modes available)
void UpdateCamera(Camera *camera);                                      // Update camera position for selected mode

void SetCameraPanControl(int panKey);                                   // Set camera pan key to combine with mouse movement (free camera)
void SetCameraAltControl(int altKey);                                   // Set camera alt key to combine with mouse movement (free camera)
void SetCameraSmoothZoomControl(int szKey);                             // Set camera smooth zoom key to combine with mouse (free camera)
void SetCameraMoveControls(int frontKey, int backKey,
                           int rightKey, int leftKey,
                           int upKey, int downKey);                     // Set camera move controls (1st person and 3rd person cameras)
```

module: shapes

```
// 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 (Vector version)
void DrawLineEx(Vector2 startPos, Vector2 endPos, float thick, Color color);                        // Draw a line defining thickness
void DrawLineBezier(Vector2 startPos, Vector2 endPos, float thick, Color color);                    // Draw a line using cubic-bezier curves in-out
void DrawLineStrip(Vector2 *points, int numPoints, Color color);                                    // Draw lines sequence
void DrawCircle(int centerX, int centerY, float radius, Color color);                               // Draw a color-filled circle
void DrawCircleSector(Vector2 center, float radius, int startAngle, int endAngle, int segments, Color color);     // Draw a piece of a circle
void DrawCircleSectorLines(Vector2 center, float radius, int startAngle, int endAngle, int segments, Color color);    // Draw circle sector outline
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 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, int startAngle, int endAngle, int segments, Color color); // Draw ring
void DrawRingLines(Vector2 center, float innerRadius, float outerRadius, int startAngle, int endAngle, int segments, Color color);    // Draw ring outline
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 pro parameters
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 custom vertex colors
void DrawRectangleLines(int posX, int posY, int width, int height, Color color);                    // Draw rectangle outline
void DrawRectangleLinesEx(Rectangle rec, int lineThick, Color color);                               // Draw rectangle outline with extended parameters
void DrawRectangleRounded(Rectangle rec, float roundness, int segments, Color color);               // Draw rectangle with rounded edges
void DrawRectangleRoundedLines(Rectangle rec, float roundness, int segments, int lineThick, Color color); // Draw rectangle with rounded edges outline
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color);                                 // Draw a color-filled triangle (vertex in counter-clockwise order!)
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color);                            // Draw triangle outline (vertex in counter-clockwise order!)
void DrawTriangleFan(Vector2 *points, int numPoints, Color color);                                  // Draw a triangle fan defined by points (first vertex is the center)
void DrawTriangleStrip(Vector2 *points, int pointsCount, Color color);                              // Draw a triangle strip defined by points
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

// Basic shapes collision detection functions
bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2);                                            // Check collision between two rectangles
bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2);         // Check collision between two circles
bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec);                          // Check collision between circle and rectangle
Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2);                                          // Get collision rectangle for two rectangles collision
bool CheckCollisionPointRec(Vector2 point, Rectangle rec);                                          // Check if point is inside rectangle
bool CheckCollisionPointCircle(Vector2 point, Vector2 center, float radius);                        // Check if point is inside circle
bool CheckCollisionPointTriangle(Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3);                // Check if point is inside a triangle
```

module: textures

```
  // Image/Texture2D data loading/unloading/saving functions
Image LoadImage(const char *fileName);                                                              // Load image from file into CPU memory (RAM)
Image LoadImageEx(Color *pixels, int width, int height);                                            // Load image from Color array data (RGBA - 32bit)
Image LoadImagePro(void *data, int width, int height, int format);                                  // Load image from raw data with parameters
Image LoadImageRaw(const char *fileName, int width, int height, int format, int headerSize);       // Load image from RAW file data
void ExportImage(Image image, const char *fileName);                                                // Export image data to file
void ExportImageAsCode(Image image, const char *fileName);                                          // Export image as code file defining an array of bytes
Texture2D LoadTexture(const char *fileName);                                                        // Load texture from file into GPU memory (VRAM)
Texture2D LoadTextureFromImage(Image image);                                                        // Load texture from image data
TextureCubemap LoadTextureCubemap(Image image, int layoutType);                                     // Load cubemap from image, multiple image cubemap layouts supported
RenderTexture2D LoadRenderTexture(int width, int height);                                           // Load texture for rendering (framebuffer)
void UnloadImage(Image image);                                                                      // Unload image from CPU memory (RAM)
void UnloadTexture(Texture2D texture);                                                              // Unload texture from GPU memory (VRAM)
void UnloadRenderTexture(RenderTexture2D target);                                                   // Unload render texture from GPU memory (VRAM)
Color *GetImageData(Image image);                                                                   // Get pixel data from image as a Color struct array
Vector4 *GetImageDataNormalized(Image image);                                                       // Get pixel data from image as Vector4 array (float normalized)
Rectangle GetImageAlphaBorder(Image image, float threshold);                                        // Get image alpha border rectangle
int GetPixelDataSize(int width, int height, int format);                                            // Get pixel data size in bytes (image or texture)
Image GetTextureData(Texture2D texture);                                                            // Get pixel data from GPU texture and return an Image
Image GetScreenData(void);                                                                          // Get pixel data from screen buffer and return an Image (screenshot)
void UpdateTexture(Texture2D texture, const void *pixels);                                          // Update GPU texture with new data

// Image manipulation functions
Image ImageCopy(Image image);                                                                       // Create an image duplicate (useful for transformations)
Image ImageFromImage(Image image, Rectangle rec);                                                   // Create an image from another image piece
void ImageToPOT(Image *image, Color fillColor);                                                     // Convert image to POT (power-of-two)
void ImageFormat(Image *image, int newFormat);                                                      // Convert image data to desired format
void ImageAlphaMask(Image *image, Image alphaMask);                                                 // Apply alpha mask to image
void ImageAlphaClear(Image *image, Color color, float threshold);                                   // Clear alpha channel to desired color
void ImageAlphaCrop(Image *image, float threshold);                                                 // Crop image depending on alpha value
void ImageAlphaPremultiply(Image *image);                                                           // Premultiply alpha channel
void ImageCrop(Image *image, Rectangle crop);                                                       // Crop an image to a defined rectangle
void ImageResize(Image *image, int newWidth, int newHeight);                                        // Resize image (Bicubic scaling algorithm)
void ImageResizeNN(Image *image, int newWidth,int newHeight);                                       // Resize image (Nearest-Neighbor scaling algorithm)
void ImageResizeCanvas(Image *image, int newWidth, int newHeight, int offsetX, int offsetY, Color color);  // Resize canvas and fill with color
void ImageMipmaps(Image *image);                                                                    // Generate all mipmap levels for a provided image
void ImageDither(Image *image, int rBpp, int gBpp, int bBpp, int aBpp);                             // Dither image data to 16bpp or lower (Floyd-Steinberg dithering)
Color *ImageExtractPalette(Image image, int maxPaletteSize, int *extractCount);                     // Extract color palette from image to maximum size (memory should be freed
Image ImageText(const char *text, int fontSize, Color color);                                       // Create an image from text (default font)
Image ImageTextEx(Font font, const char *text, float fontSize, float spacing, Color tint);          // Create an image from text (custom sprite font)
void ImageDraw(Image *dst, Image src, Rectangle srcRec, Rectangle dstRec, Color tint);              // Draw a source image within a destination image (tint applied to source)
void ImageDrawRectangle(Image *dst, Rectangle rec, Color color);                                    // Draw rectangle within an image
```

```
void ImageDrawRectangleLines(Image *dst, Rectangle rec, int thick, Color color);                    // Draw rectangle lines within an image
void ImageDrawText(Image *dst, Vector2 position, const char *text, int fontSize, Color color);      // Draw text (default font) within an image (destination)
void ImageDrawTextEx(Image *dst, Vector2 position, Font font, const char *text, float fontSize, float spacing, Color color); // Draw text (custom sprite font) within an image
void ImageFlipVertical(Image *image);                                                               // Flip image vertically
void ImageFlipHorizontal(Image *image);                                                             // Flip image horizontally
void ImageRotateCW(Image *image);                                                                   // Rotate image clockwise 90deg
void ImageRotateCCW(Image *image);                                                                  // Rotate image counter-clockwise 90deg
void ImageColorTint(Image *image, Color color);                                                     // Modify image color: tint
void ImageColorInvert(Image *image);                                                                // Modify image color: invert
void ImageColorGrayscale(Image *image);                                                             // Modify image color: grayscale
void ImageColorContrast(Image *image, float contrast);                                              // Modify image color: contrast (-100 to 100)
void ImageColorBrightness(Image *image, int brightness);                                            // Modify image color: brightness (-255 to 255)
void ImageColorReplace(Image *image, Color color, Color replace);                                   // Modify image color: replace color

// Image generation functions
Image GenImageColor(int width, int height, Color color);                                            // Generate image: plain color
Image GenImageGradientV(int width, int height, Color top, Color bottom);                            // Generate image: vertical gradient
Image GenImageGradientH(int width, int height, Color left, Color right);                            // Generate image: horizontal gradient
Image GenImageGradientRadial(int width, int height, float density, Color inner, Color outer);      // Generate image: radial gradient
Image GenImageChecked(int width, int height, int checksX, int checksY, Color col1, Color col2);     // Generate image: checked
Image GenImageWhiteNoise(int width, int height, float factor);                                      // Generate image: white noise
Image GenImagePerlinNoise(int width, int height, int offsetX, int offsetY, float scale);            // Generate image: perlin noise
Image GenImageCellular(int width, int height, int tileSize);                                        // Generate image: cellular algorithm. Bigger tileSize means bigger cells

// Texture2D configuration functions
void GenTextureMipmaps(Texture2D *texture);                                                         // Generate GPU mipmaps for a texture
void SetTextureFilter(Texture2D texture, int filterMode);                                           // Set texture scaling filter mode
void SetTextureWrap(Texture2D texture, int wrapMode);                                               // Set texture wrapping mode

// Texture2D configuration functions
void GenTextureMipmaps(Texture2D *texture);                                                         // Generate GPU mipmaps for a texture
void SetTextureFilter(Texture2D texture, int filterMode);                                           // Set texture scaling filter mode
void SetTextureWrap(Texture2D texture, int wrapMode);                                               // Set texture wrapping mode

// Texture2D drawing functions
void DrawTexture(Texture2D texture, int posX, int posY, Color tint);                                // Draw a Texture2D
void DrawTextureV(Texture2D texture, Vector2 position, Color tint);                                 // Draw a Texture2D with position defined as Vector2
void DrawTextureEx(Texture2D texture, Vector2 position, float rotation, float scale, Color tint);   // Draw a Texture2D with extended parameters
void DrawTextureRec(Texture2D texture, Rectangle sourceRec, Vector2 position, Color tint);          // Draw a part of a texture defined by a rectangle
void DrawTextureQuad(Texture2D texture, Vector2 tiling, Vector2 offset, Rectangle quad, Color tint);  // Draw texture quad with tiling and offset parameters
void DrawTexturePro(Texture2D texture, Rectangle sourceRec, Rectangle destRec, Vector2 origin, float rotation, Color tint);       // Draw a part of a texture defined by a rect
void DrawTextureNPatch(Texture2D texture, NPatchInfo nPatchInfo, Rectangle destRec, Vector2 origin, float rotation, Color tint);  // Draws a texture (or part of it) that stret
```

module: text

```
// 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 *fontChars, int charsCount);                // Load font from file with extended parameters
Font LoadFontFromImage(Image image, Color key, int firstChar);                                      // Load font from Image (XNA style)
CharInfo *LoadFontData(const char *fileName, int fontSize, int *fontChars, int charsCount, int type); // Load font data for further use
Image GenImageFontAtlas(const CharInfo *chars, Rectangle **recs, int charsCount, int fontSize, int padding, int packMethod);  // Generate image font atlas using chars info
void UnloadFont(Font font);                                                                         // Unload Font from GPU memory (VRAM)

// Text drawing functions
void DrawFPS(int posX, int posY);                                                                   // Shows 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 DrawTextRec(Font font, const char *text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint);   // Draw text using font inside rectangle limits
void DrawTextRecEx(Font font, const char *text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint,
                   int selectStart, int selectLength, Color selectTint, Color selectBackTint);      // Draw text using font inside rectangle limits with support for text selecti
void DrawTextCodepoint(Font font, int codepoint, Vector2 position, float scale, Color tint);        // Draw one character (codepoint)

// Text misc. functions
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 index position for a unicode character on font

// Text strings management functions (no utf8 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 string 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 (memory must be freed!)
char *TextInsert(const char *text, const char *insert, int position);                               // Insert text in a position (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)
char *TextToUtf8(int *codepoints, int length);                                                      // Encode text codepoint into utf8 text (memory must be freed!)

// UTF8 text strings management functions
int *GetCodepoints(const char *text, int *count);                                                   // Get all codepoints in a string, codepoints count returned by parameters
int GetCodepointsCount(const char *text);                                                           // Get total number of characters (codepoints) in a UTF8 encoded string
int GetNextCodepoint(const char *text, int *bytesProcessed);                                       // Returns next codepoint in a UTF8 encoded string; 0x3f('?') is returned on
const char *CodepointToUtf8(int codepoint, int *byteLength);                                        // Encode codepoint into utf8 text (char array length returned as parameter)
```

module: models

```
  // 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 line
void DrawCircle3D(Vector3 center, float radius, Vector3 rotationAxis, float rotationAngle, Color color); // Draw a circle in 3D world space
void DrawCube(Vector3 position, float width, float height, float length, Color color);              // Draw cube
void DrawCubeV(Vector3 position, Vector3 size, Color color);                                        // Draw cube (Vector version)
void DrawCubeWires(Vector3 position, float width, float height, float length, Color color);        // Draw cube wires
void DrawCubeWiresV(Vector3 position, Vector3 size, Color color);                                   // Draw cube wires (Vector version)
void DrawCubeTexture(Texture2D texture, Vector3 position, float width, float height, float length, Color color); // Draw cube textured
void DrawSphere(Vector3 centerPos, float radius, Color color);                                      // Draw sphere
void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color color);            // Draw sphere with extended parameters
void DrawSphereWires(Vector3 centerPos, float radius, int rings, int slices, Color color);         // Draw sphere wires
void DrawCylinder(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone
void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone wires
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))
void DrawGizmo(Vector3 position);                                                                   // Draw simple gizmo

// Model loading/unloading functions
Model LoadModel(const char *fileName);                                                              // Load model from files (meshes and materials)
Model LoadModelFromMesh(Mesh mesh);                                                                 // Load model from generated mesh (default material)
void UnloadModel(Model model);                                                                      // Unload model from memory (RAM and/or VRAM)

// Mesh loading/unloading functions
Mesh *LoadMeshes(const char *fileName, int *meshCount);                                             // Load meshes from model file
void ExportMesh(Mesh mesh, const char *fileName);                                                   // Export mesh data to file
void UnloadMesh(Mesh mesh);                                                                         // Unload mesh from memory (RAM and/or VRAM)

// 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, NORMAL maps)
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 (MAP_DIFFUSE, MAP_SPECULAR...)
void SetModelMeshMaterial(Model *model, int meshId, int materialId);                                // Set material for a mesh

// Model animations loading/unloading functions
ModelAnimation *LoadModelAnimations(const char *fileName, int *animsCount);                         // Load model animations from file
void UpdateModelAnimation(Model model, ModelAnimation anim, int frame);                             // Update model animation pose
```

```
void UnloadModelAnimation(ModelAnimation anim);                                 // Unload animation data
bool IsModelAnimationValid(Model model, ModelAnimation anim);                   // Check model animation skeleton match

// 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 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 GenMeshCubicmap(Image cubicmap, Vector3 cubeSize);                         // Generate cubes-based map mesh from image data

// Mesh manipulation functions
BoundingBox MeshBoundingBox(Mesh mesh);                                         // Compute mesh bounding box limits
void MeshTangents(Mesh *mesh);                                                  // Compute mesh tangents
void MeshBinormals(Mesh *mesh);                                                 // Compute mesh binormals

// 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 extended parameters
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 wires (with texture if set) with ex
void DrawBoundingBox(BoundingBox box, Color color);                                                     // Draw bounding box (wires)
void DrawBillboard(Camera camera, Texture2D texture, Vector3 center, float size, Color tint);           // Draw a billboard texture
void DrawBillboardRec(Camera camera, Texture2D texture, Rectangle sourceRec, Vector3 center, float size, Color tint); // Draw a billboard texture defined by sourceRec

// Collision detection functions
bool CheckCollisionSpheres(Vector3 centerA, float radiusA, Vector3 centerB, float radiusB);       // Detect collision between two spheres
bool CheckCollisionBoxes(BoundingBox box1, BoundingBox box2);                                     // Detect collision between two bounding boxes
bool CheckCollisionBoxSphere(BoundingBox box, Vector3 center, float radius);                      // Detect collision between box and sphere
bool CheckCollisionRaySphere(Ray ray, Vector3 center, float radius);                              // Detect collision between ray and sphere
bool CheckCollisionRaySphereEx(Ray ray, Vector3 center, float radius, Vector3 *collisionPoint);   // Detect collision between ray and sphere, returns collision point
bool CheckCollisionRayBox(Ray ray, BoundingBox box);                                              // Detect collision between ray and box
RayHitInfo GetCollisionRayModel(Ray ray, Model model);                                            // Get collision info between ray and model
RayHitInfo GetCollisionRayTriangle(Ray ray, Vector3 p1, Vector3 p2, Vector3 p3);                  // Get collision info between ray and triangle
RayHitInfo GetCollisionRayGround(Ray ray, float groundHeight);                                    // Get collision info between ray and ground plane (Y-normal plane)
```

module: shaders (rlgl)

```
// Shader loading/unloading functions
char *LoadText(const char *fileName);                                           // Load chars array from text file
Shader LoadShader(const char *vsFileName, const char *fsFileName);              // Load shader from files and bind default locations
Shader LoadShaderCode(char *vsCode, char *fsCode);                              // Load shader from code strings and bind default locations
void UnloadShader(Shader shader);                                               // Unload shader from GPU memory (VRAM)

Shader GetShaderDefault(void);                                                  // Get default shader
Texture2D GetTextureDefault(void);                                              // Get default texture
Texture2D GetShapesTexture(void);                                               // Get texture to draw shapes
Rectangle GetShapesTextureRec(void);                                            // Get texture rectangle to draw shapes
void SetShapesTexture(Texture2D texture, Rectangle source);                     // Define default texture used to draw shapes

// Shader configuration functions
int GetShaderLocation(Shader shader, const char *uniformName);                                  // Get shader uniform location
void SetShaderValue(Shader shader, int uniformLoc, const void *value, int uniformType);        // Set shader uniform value
void SetShaderValueV(Shader shader, int uniformLoc, const void *value, int uniformType, int count); // Set shader uniform value vector
void SetShaderValueMatrix(Shader shader, int uniformLoc, Matrix mat);                           // Set shader uniform value (matrix 4x4)
void SetShaderValueTexture(Shader shader, int uniformLoc, Texture2D texture);                   // Set shader uniform value for texture
void SetMatrixProjection(Matrix proj);                                                          // Set a custom projection matrix (replaces internal projection matrix)
void SetMatrixModelview(Matrix view);                                                           // Set a custom modelview matrix (replaces internal modelview matrix)
Matrix GetMatrixModelview();                                                                    // Get internal modelview matrix
Matrix GetMatrixProjection(void);                                                               // Get internal projection matrix

// Shading begin/end functions
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)
void EndBlendMode(void);                                                        // End blending mode (reset to default: alpha blending)

// VR control functions
void InitVrSimulator(void);                                                     // Init VR simulator for selected device parameters
void CloseVrSimulator(void);                                                    // Close VR simulator for current device
void UpdateVrTracking(Camera *camera);                                          // Update VR tracking (position and orientation) and camera
void SetVrConfiguration(VrDeviceInfo info, Shader distortion);                  // Set stereo rendering configuration parameters
bool IsVrSimulatorReady(void);                                                  // Detect if VR simulator is ready
void ToggleVrMode(void);                                                        // Enable/Disable VR experience
void BeginVrDrawing(void);                                                      // Begin VR simulator stereo rendering
void EndVrDrawing(void);                                                        // End VR simulator stereo rendering
```

module: audio

```
// Audio device management functions
void InitAudioDevice(void);                                     // Initialize audio device and context
void CloseAudioDevice(void);                                    // Close the audio device and context (and music stream)
bool IsAudioDeviceReady(void);                                  // Check if audio device is ready
void SetMasterVolume(float volume);                             // Set master volume (listener)

// Wave/Sound loading/unloading functions
Wave LoadWave(const char *fileName);                            // Load wave data from file
Wave LoadWaveEx(void *data, int sampleCount, int sampleRate, int sampleSize, int channels); // Load wave data from raw array data
Sound LoadSound(const char *fileName);                          // Load sound from file
Sound LoadSoundFromWave(Wave wave);                             // Load sound from wave data
void UpdateSound(Sound sound, const void *data, int samplesCount);              // Update sound buffer with new data
void UnloadWave(Wave wave);                                     // Unload wave data
void UnloadSound(Sound sound);                                  // Unload sound
void ExportWave(Wave wave, const char *fileName);               // Export wave data to file
void ExportWaveAsCode(Wave wave, const char *fileName);         // Export wave sample data to code (.h)

// Wave/Sound management functions
void PlaySound(Sound sound);                                    // Play a sound
void StopSound(Sound sound);                                    // Stop playing a sound
void PauseSound(Sound sound);                                   // Pause a sound
void ResumeSound(Sound sound);                                  // Resume a paused sound
void PlaySoundMulti(Sound sound);                               // Play a sound (using multichannel buffer pool)
void StopSoundMulti(void);                                      // Stop any sound playing (using multichannel buffer pool)
int GetSoundsPlaying(void);                                     // Get number of sounds playing in the multichannel
bool IsSoundPlaying(Sound sound);                               // Check if a sound is currently playing
void SetSoundVolume(Sound sound, float volume);                 // Set volume for a sound (1.0 is max level)
void SetSoundPitch(Sound sound, float pitch);                   // Set pitch for a sound (1.0 is base level)
void WaveFormat(Wave *wave, int sampleRate, int sampleSize, int channels);      // Convert wave data to desired format
Wave WaveCopy(Wave wave);                                       // Copy a wave to a new wave
void WaveCrop(Wave *wave, int initSample, int finalSample);     // Crop a wave to defined samples range
float *GetWaveData(Wave wave);                                  // Get samples data from wave as a floats array

// Music management functions
Music LoadMusicStream(const char *fileName);                    // Load music stream from file
void UnloadMusicStream(Music music);                            // Unload music stream
void PlayMusicStream(Music music);                              // Start music playing
void UpdateMusicStream(Music music);                            // Updates buffers for music streaming
void StopMusicStream(Music music);                              // Stop music playing
void PauseMusicStream(Music music);                             // Pause music playing
void ResumeMusicStream(Music music);                            // Resume playing paused music
bool IsMusicPlaying(Music music);                               // Check if music is playing
void SetMusicVolume(Music music, float volume);                 // Set volume for music (1.0 is max level)
void SetMusicPitch(Music music, float pitch);                   // Set pitch for a music (1.0 is base level)
void SetMusicLoopCount(Music music, int count);                 // Set music loop count (loop repeats)
float GetMusicTimeLength(Music music);                          // Get music time length (in seconds)
float GetMusicTimePlayed(Music music);                          // Get current music time played (in seconds)

// AudioStream management functions
AudioStream InitAudioStream(unsigned int sampleRate, unsigned int sampleSize, unsigned int channels); // Init audio stream (to stream raw audio pcm data)
void UpdateAudioStream(AudioStream stream, const void *data, int samplesCount); // Update audio stream buffers with data
void CloseAudioStream(AudioStream stream);                      // Close audio stream and free memory
bool IsAudioBufferProcessed(AudioStream stream);                // Check if any audio stream buffers requires refill
```

```
void PlayAudioStream(AudioStream stream);                 // Play audio stream
void PauseAudioStream(AudioStream stream);                // Pause audio stream
void ResumeAudioStream(AudioStream stream);               // Resume audio stream
bool IsAudioStreamPlaying(AudioStream stream);            // Check if audio stream is playing
void StopAudioStream(AudioStream stream);                 // Stop audio stream
void SetAudioStreamVolume(AudioStream stream, float volume);  // Set volume for audio stream (1.0 is max level)
void SetAudioStreamPitch(AudioStream stream, float pitch);    // Set pitch for audio stream (1.0 is base level)
```

structs

```
struct Vector2;          // Vector2 type
struct Vector3;          // Vector3 type
struct Vector4;          // Vector4 type
struct Quaternion;       // Quaternion type
struct Matrix;           // Matrix type (OpenGL style 4x4)
struct Color;            // Color type, RGBA (32bit)
struct Rectangle;        // Rectangle type

struct Image;            // Image type (multiple pixel formats supported)
                         // NOTE: Data stored in CPU memory (RAM)
struct Texture;          // Texture type (multiple internal formats supported)
                         // NOTE: Data stored in GPU memory (VRAM)
struct RenderTexture;    // RenderTexture type, for texture rendering
struct NPatchInfo;       // N-Patch layout info
struct CharInfo;         // Font character info
struct Font;             // Font type, includes texture and chars data

struct Camera;           // Camera type, defines 3d camera position/orientation
struct Camera2D;         // Camera2D type, defines a 2d camera
struct Mesh;             // Vertex data definning a mesh
struct Shader;           // Shader type (generic shader)
struct MaterialMap;      // Material texture map
struct Material;         // Material type
struct Model;            // Basic 3d Model type
struct Transform;        // Transformation (used for bones)
struct BoneInfo;         // Bone information
struct ModelAnimation;   // Model animation data (bones and frames)
struct Ray;              // Ray type (useful for raycast)
struct RayHitInfo;       // Raycast hit information
struct BoundingBox;      // Bounding box type for 3d mesh

struct Wave;             // Wave type, defines audio wave data
struct Sound;            // Basic Sound source and buffer
struct Music;            // Music type (file streaming from memory)
struct AudioStream;      // Raw audio stream type

struct VrDeviceInfo;     // VR device parameters
```

colors

```
// Custom raylib color palette for amazing visuals
#define LIGHTGRAY  (Color){ 200, 200, 200, 255 }   // Light Gray
#define GRAY       (Color){ 130, 130, 130, 255 }   // Gray
#define DARKGRAY   (Color){ 80, 80, 80, 255 }      // Dark Gray
#define YELLOW     (Color){ 253, 249, 0, 255 }     // Yellow
#define GOLD       (Color){ 255, 203, 0, 255 }     // Gold
#define ORANGE     (Color){ 255, 161, 0, 255 }     // Orange
#define PINK       (Color){ 255, 109, 194, 255 }   // Pink
#define RED        (Color){ 230, 41, 55, 255 }     // Red
#define MAROON     (Color){ 190, 33, 55, 255 }     // Maroon
#define GREEN      (Color){ 0, 228, 48, 255 }      // Green
#define LIME       (Color){ 0, 158, 47, 255 }      // Lime
#define DARKGREEN  (Color){ 0, 117, 44, 255 }      // Dark Green
#define SKYBLUE    (Color){ 102, 191, 255, 255 }   // Sky Blue
#define BLUE       (Color){ 0, 121, 241, 255 }     // Blue
#define DARKBLUE   (Color){ 0, 82, 172, 255 }      // Dark Blue
#define PURPLE     (Color){ 200, 122, 255, 255 }   // Purple
#define VIOLET     (Color){ 135, 60, 190, 255 }    // Violet
#define DARKPURPLE (Color){ 112, 31, 126, 255 }    // Dark Purple
#define BEIGE      (Color){ 211, 176, 131, 255 }   // Beige
#define BROWN      (Color){ 127, 106, 79, 255 }    // Brown
#define DARKBROWN  (Color){ 76, 63, 47, 255 }      // Dark Brown

#define WHITE      (Color){ 255, 255, 255, 255 }   // White
#define BLACK      (Color){ 0, 0, 0, 255 }         // Black
#define BLANK      (Color){ 0, 0, 0, 0 }           // Transparent
#define MAGENTA    (Color){ 255, 0, 255, 255 }     // Magenta
#define RAYWHITE   (Color){ 245, 245, 245, 255 }   // Ray White
```