

## module: core

```
// Window-related functions
void InitWindow(int width, int height, const char *title);
bool WindowShouldClose(void);
void CloseWindow(void);
bool IsWindowReady(void);
bool IsWindowFullscreen(void);
bool IsWindowHidden(void);
bool IsWindowMinimized(void);
bool IsWindowMaximized(void);
bool IsWindowFocused(void);
bool IsWindowResized(void);
bool IsWindowState(unsigned int flag);
void SetWindowState(unsigned int flags);
void ClearWindowState(unsigned int flags);
void ToggleFullscreen(void);
void MaximizeWindow(void);
void MinimizeWindow(void);
void RestoreWindow(void);
void SetWindowIcon(Image image);
void SetWindowTitle(const char *title);
void SetWindowPosition(int x, int y);
void SetWindowMonitor(int monitor);
void SetWindowMinSize(int width, int height);
void SetWindowSize(int width, int height);
void *GetWindowHandle(void);
int GetScreenWidth(void);
int GetScreenHeight(void);
int GetMonitorCount(void);
Vector2 GetMonitorPosition(int monitor);
int GetMonitorWidth(int monitor);
int GetMonitorHeight(int monitor);
int GetMonitorPhysicalWidth(int monitor);
int GetMonitorPhysicalHeight(int monitor);
int GetMonitorRefreshRate(int monitor);
Vector2 GetWindowPosition(void);
Vector2 GetWindowScaleDPI(void);
const char *GetMonitorName(int monitor);
void SetClipboardText(const char *text);
const char *GetClipboardText(void);

// Cursor-related functions
void ShowCursor(void);
void HideCursor(void);
bool IsCursorHidden(void);
void EnableCursor(void);
void DisableCursor(void);
bool IsCursorOnScreen(void);

// Drawing-related functions
void ClearBackground(Color color);
void BeginDrawing(void);
void EndDrawing(void);
void BeginMode2D(Camera2D camera);
void EndMode2D(void);
void BeginMode3D(Camera3D camera);
void EndMode3D(void);
void BeginTextureMode(RenderTexture2D target);
void EndTextureMode(void);
void BeginScissorMode(int x, int y, int width, int height);
void EndScissorMode(void);

// Screen-space-related functions
Ray GetMouseRay(Vector2 mousePosition, Camera camera);
Matrix GetCameraMatrix(Camera camera);
Matrix GetCameraMatrix2D(Camera2D camera);
Vector2 GetWorldToScreen(Vector3 position, Camera camera);
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);
Vector2 GetScreenToWorld2D(Vector2 position, Camera2D camera);

// Timing-related functions
void SetTargetFPS(int fps);
int GetFPS(void);
float GetFrameTime(void);
double GetTime(void);

// Misc. functions
void SetConfigFlags(unsigned int flags);

void SetTraceLogLevel(int logType);
void SetTraceLogExit(int logType);
void SetTraceLogCallback(TraceLogCallback callback);
void TraceLog(int logType, const char *text, ...);

void *MemAlloc(int size);
void MemFree(void *ptr);
void TakeScreenshot(const char *fileName);
int GetRandomValue(int min, int max);

// Files management functions
unsigned char *LoadFileData(const char *fileName, unsigned int *bytesRead); // 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, unsigned int bytesToWrite); // Save data to file from byte array (write), returns true on success
char *LoadFileText(const char *fileName); // Load text data from file (read), returns a '\0' terminated string
void UnloadFileText(unsigned 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 on success
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)
const char *GetFileExtension(const char *fileName); // Get pointer to extension for a filename string (including point: ".png")
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, return true on 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 algorithm)
unsigned char *DecompressData(unsigned char *compData, int compDataLength, int *dataLength); // Decompress data (DEFLATE algorithm)

// Persistent storage management
bool SaveStorageValue(unsigned int position, int value); // Save integer value to storage file (to defined position), returns true on success
int LoadStorageValue(unsigned 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 (Module: core)
```

```
//-----

// Input-related functions: keyboard
bool IsKeyPressed(int key);
bool IsKeyDown(int key);
bool IsKeyReleased(int key);
bool IsKeyUp(int key);
void SetExitKey(int key);
int GetKeyPressed(void);
int GetCharPressed(void);

// Input-related functions: gamepads
bool IsGamepadAvailable(int gamepad);
bool IsGamepadName(int gamepad, const char *name);
const char *GetGamepadName(int gamepad);
bool IsGamepadButtonPressed(int gamepad, int button);
bool IsGamepadButtonDown(int gamepad, int button);
bool IsGamepadButtonReleased(int gamepad, int button);
bool IsGamepadButtonUp(int gamepad, int button);
int GetGamepadButtonPressed(void);
int GetGamepadAxisCount(int gamepad);
float GetGamepadAxisMovement(int gamepad, int axis);

// Input-related functions: mouse
bool IsMouseButtonPressed(int button);
bool IsMouseButtonDown(int button);
bool IsMouseButtonReleased(int button);
bool IsMouseButtonUp(int button);
int GetMouseX(void);
int GetMouseY(void);
Vector2 GetMousePosition(void);
void SetMousePosition(int x, int y);
void SetMouseOffset(int offsetX, int offsetY);
void SetMouseScale(float scaleX, float scaleY);
float GetMouseWheelMove(void);
int GetMouseCursor(void);
void SetMouseCursor(int cursor);

// Input-related functions: touch
int GetTouchX(void);
int GetTouchY(void);
Vector2 GetTouchPosition(int index);

// Detect if a key has been pressed once
// Detect if a key is being pressed
// Detect if a key has been released once
// Detect if a key is NOT being pressed
// Set a custom key to exit program (default is ESC)
// Get key pressed (keycode), call it multiple times for keys queued
// Get char pressed (unicode), call it multiple times for chars queued

// Detect if a gamepad is available
// Check gamepad name (if available)
// Return gamepad internal name id
// Detect if a gamepad button has been pressed once
// Detect if a gamepad button is being pressed
// Detect if a gamepad button has been released once
// Detect if a gamepad button is NOT being pressed
// Get the last gamepad button pressed
// Return gamepad axis count for a gamepad
// Return axis movement value for a gamepad axis

// Detect if a mouse button has been pressed once
// Detect if a mouse button is being pressed
// Detect if a mouse button has been released once
// Detect if a mouse button is NOT being pressed
// Returns mouse position X
// Returns mouse position Y
// Returns mouse position XY
// Set mouse position XY
// Set mouse offset
// Set mouse scaling
// Returns mouse wheel movement Y
// Returns mouse cursor if (MouseCursor enum)
// Set mouse cursor

// Returns touch position X for touch point 0 (relative to screen size)
// Returns touch position Y for touch point 0 (relative to screen size)
// 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);
bool IsGestureDetected(int gesture);
int GetGestureDetected(void);
int GetTouchPointsCount(void);
float GetGestureHoldDuration(void);
Vector2 GetGestureDragVector(void);
float GetGestureDragAngle(void);
Vector2 GetGesturePinchVector(void);
float GetGesturePinchAngle(void);

// Enable a set of gestures using flags
// Check if a gesture have been detected
// Get latest detected gesture
// Get touch points count
// Get gesture hold time in milliseconds
// Get gesture drag vector
// Get gesture drag angle
// Get gesture pinch delta
// Get gesture pinch angle

//-----
// Camera System Functions (Module: camera)
//-----
void SetCameraMode(Camera camera, int mode);
void UpdateCamera(Camera *camera);

// Set camera mode (multiple camera modes available)
// Update camera position for selected mode

void SetCameraPanControl(int keyPan);
void SetCameraAltControl(int keyAlt);
void SetCameraSmoothZoomControl(int keySmoothZoom);
void SetCameraMoveControls(int frontKey, int backKey,
                           int rightKey, int leftKey,
                           int upKey, int downKey);

// Set camera pan key to combine with mouse movement (free camera)
// Set camera alt key to combine with mouse movement (free camera)
// Set camera smooth zoom key to combine with mouse (free camera)

// Set camera move controls (1st person and 3rd person cameras)
```

## module: shapes

```
// Basic shapes drawing functions
void DrawPixel(int posX, int posY, Color color);
void DrawPixelV(Vector2 position, Color color);
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color);
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color);
void DrawLineEx(Vector2 startPos, Vector2 endPos, float thick, Color color);
void DrawLineBezier(Vector2 startPos, Vector2 endPos, float thick, Color color);
void DrawLineStrip(Vector2 *points, int pointsCount, Color color);
void DrawCircle(int centerX, int centerY, float radius, Color color);
void DrawCircleSector(Vector2 center, float radius, int startAngle, int endAngle, int segments, Color color);
void DrawCircleSectorLines(Vector2 center, float radius, int startAngle, int endAngle, int segments, Color color);
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2);
void DrawCircleV(Vector2 center, float radius, Color color);
void DrawCircleLines(int centerX, int centerY, float radius, Color color);
void DrawEllipse(int centerX, int centerY, float radiusH, float radiusV, Color color);
void DrawEllipseLines(int centerX, int centerY, float radiusH, float radiusV, Color color);
void DrawRing(Vector2 center, float innerRadius, float outerRadius, int startAngle, int endAngle, int segments, Color color);
void DrawRingLines(Vector2 center, float innerRadius, float outerRadius, int startAngle, int endAngle, int segments, Color color);
void DrawRectangle(int posX, int posY, int width, int height, Color color);
void DrawRectangleV(Vector2 position, Vector2 size, Color color);
void DrawRectangleRec(Rectangle rec, Color color);
void DrawRectanglePro(Rectangle rec, Vector2 origin, float rotation, Color color);
void DrawRectangleGradientV(int posX, int posY, int width, int height, Color color1, Color color2);
void DrawRectangleGradientH(int posX, int posY, int width, int height, Color color1, Color color2);
void DrawRectangleGradientEx(Rectangle rec, Color col1, Color col2, Color col3, Color col4);
void DrawRectangleLines(int posX, int posY, int width, int height, Color color);
void DrawRectangleLinesEx(Rectangle rec, int lineThick, Color color);
void DrawRectangleRounded(Rectangle rec, float roundness, int segments, Color color);
void DrawRectangleRoundedLines(Rectangle rec, float roundness, int segments, int lineThick, Color color);
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color);
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color);
void DrawTriangleFan(Vector2 *points, int pointsCount, Color color);
void DrawTriangleStrip(Vector2 *points, int pointsCount, Color color);
void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color);
void DrawPolyLines(Vector2 center, int sides, float radius, float rotation, Color color);

// Draw a pixel
// Draw a pixel (Vector version)
// Draw a line
// Draw a line (Vector version)
// Draw a line defining thickness
// Draw a line using cubic-bezier curves in-out
// Draw lines sequence
// Draw a color-filled circle
// Draw a piece of a circle
// Draw circle sector outline
// Draw a gradient-filled circle
// Draw a color-filled circle (Vector version)
// Draw circle outline
// Draw ellipse
// Draw ellipse outline
// Draw ring
// Draw ring outline
// Draw a color-filled rectangle
// Draw a color-filled rectangle (Vector version)
// Draw a color-filled rectangle
// Draw a color-filled rectangle with pro parameters
// Draw a vertical-gradient-filled rectangle
// Draw a horizontal-gradient-filled rectangle
// Draw a gradient-filled rectangle with custom vertex colors
// Draw rectangle outline
// Draw rectangle outline with extended parameters
// Draw rectangle with rounded edges
// Draw rectangle with rounded edges outline
// Draw a color-filled triangle (vertex in counter-clockwise order!)
// Draw triangle outline (vertex in counter-clockwise order!)
// Draw a triangle fan defined by points (first vertex is the center)
// Draw a triangle strip defined by points
// Draw a regular polygon (Vector version)
// Draw a polygon outline of n sides

// 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 CheckCollisionLines(Vector2 startPos1, Vector2 endPos1, Vector2 startPos2, Vector2 endPos2, Vector2 *collisionPoint);
Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2);

// Check collision between two rectangles
// Check collision between two circles
// Check collision between circle and rectangle
// Check if point is inside rectangle
// Check if point is inside circle
// Check if point is inside a triangle
// Check the collision between two lines
// Get collision rectangle for two rectangles collision
```

## module: textures

```
// Image loading functions
// NOTE: This 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 LoadImageAnim(const char *fileName, int *frames);
Image LoadImageFromMemory(const char *fileType, const unsigned char *fileData, int dataSize);
void UnloadImage(Image image);
bool ExportImage(Image image, const char *fileName);
bool ExportImageAsCode(Image image, const char *fileName);

// Load image from file into CPU memory (RAM)
// Load image from RAW file data
// Load image sequence from file (frames appended to image.data)
// Load image from memory buffer, fileType refers to extension: i.e. "png"
// Unload image from CPU memory (RAM)
// Export image data to file, returns true on success
// Export image as code file defining an array of bytes, returns true on suc

// 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);
Image GenImageGradientH(int width, int height, Color left, Color right);
Image GenImageGradientRadial(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 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 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 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 *colorsCount);
void UnloadImageColors(Color *colors);
void UnloadImagePalette(Color *colors);
Rectangle GetImageAlphaBorder(Image image, float threshold);

// 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 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);
void ImageDrawTextEx(Image *dst, Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint);

// Texture loading functions
// NOTE: These functions require GPU access
Texture2D LoadTexture(const char *fileName);
Texture2D LoadTextureFromImage(Image image);
TextureCubemap LoadTextureCubemap(Image image, int layoutType);
RenderTexture2D LoadRenderTexture(int width, int height);
void UnloadTexture(Texture2D texture);
void UnloadRenderTexture(RenderTexture2D target);
void UpdateTexture(Texture2D texture, const void *pixels);
void UpdateTextureRec(Texture2D texture, Rectangle rec, const void *pixels);
Image GetTextureData(Texture2D texture);
Image GetScreenData(void);

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

// 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 DrawTextureQuad(Texture2D texture, Vector2 tiling, Vector2 offset, Rectangle quad, Color tint);
void DrawTextureTiled(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, float scale, 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 ColorAlpha(Color color, float alpha);
Color ColorAlphaBlend(Color dst, Color src, Color tint);
Color GetColor(int hexValue);
Color GetPixelColor(void *srcPtr, int format);
void SetPixelColor(void *dstPtr, Color color, int format);
int GetPixelDataSize(int width, int height, int format);

// Generate image: vertical gradient
// Generate image: horizontal gradient
// Generate image: radial gradient
// Generate image: checked
// Generate image: white noise
// Generate image: perlin noise
// Generate image: cellular algorithm. Bigger tileSize means bigger cells

// Create an image duplicate (useful for transformations)
// Create an image from another image piece
// Create an image from text (default font)
// Create an image from text (custom sprite font)
// Convert image data to desired format
// Convert image to POT (power-of-two)
// Crop an image to a defined rectangle
// Crop image depending on alpha value
// Clear alpha channel to desired color
// Apply alpha mask to image
// Premultiply alpha channel
// Resize image (Bicubic scaling algorithm)
// Resize image (Nearest-Neighbor scaling algorithm)
// Resize canvas and fill with color
// Generate all mipmap levels for a provided image
// Dither image data to 16bpp or lower (Floyd-Steinberg dithering)
// Flip image vertically
// Flip image horizontally
// Rotate image clockwise 90deg
// Rotate image counter-clockwise 90deg
// Modify image color: tint
// Modify image color: invert
// Modify image color: grayscale
// Modify image color: contrast (-100 to 100)
// Modify image color: brightness (-255 to 255)
// Modify image color: replace color
// Load color data from image as a Color array (RGBA - 32bit)
// Load colors palette from image as a Color array (RGBA - 32bit)
// Unload color data loaded with LoadImageColors()
// Unload colors palette loaded with LoadImagePalette()
// Get image alpha border rectangle

// Clear image background with given color
// Draw pixel within an image
// Draw pixel within an image (Vector version)
// Draw line within an image
// Draw line within an image (Vector version)
// Draw circle within an image
// Draw circle within an image (Vector version)
// Draw rectangle within an image
// Draw rectangle within an image (Vector version)
// Draw rectangle within an image
// Draw rectangle lines within an image
// Draw a source image within a destination image (tint applied to source)
// Draw text (using default font) within an image (destination)
// Draw text (custom sprite font) within an image (destination)

// Load texture from file into GPU memory (VRAM)
// Load texture from image data
// Load cubemap from image, multiple image cubemap layouts supported
// Load texture for rendering (framebuffer)
// Unload texture from GPU memory (VRAM)
// Unload render texture from GPU memory (VRAM)
// Update GPU texture with new data
// Update GPU texture rectangle with new data
// Get pixel data from GPU texture and return an Image
// Get pixel data from screen buffer and return an Image (screenshot)

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

// Draw a Texture2D
// Draw a Texture2D with position defined as Vector2
// Draw a Texture2D with extended parameters
// Draw a part of a texture defined by a rectangle
// Draw texture quad with tiling and offset parameters
// Draw part of a texture (defined by a rectangle)
// Draw a part of a texture defined by a rectangle
// Draws a texture (or part of it) that stretches to width and height

// Returns color with alpha applied, alpha goes from 0.0f to 1.0f
// Returns hexadecimal value for a Color
// Returns Color normalized as float [0..1]
// Returns Color from normalized values [0..1]
// Returns HSV values for a Color
// Returns a Color from HSV values
// Returns color with alpha applied, alpha goes from 0.0f to 1.0f
// Returns 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: text

```

// Font loading/unloading functions
Font GetFontDefault(void);
Font LoadFont(const char *fileName);
Font LoadFontEx(const char *fileName, int fontSize, int *fontChars, int charsCount);
Font LoadFontFromImage(Image image, Color key, int firstChar);
Font LoadFontFromMemory(const char *fileType, const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount);
CharInfo *LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount, int type);
Image GenImageFontAtlas(const CharInfo *chars, Rectangle **recs, int charsCount, int fontSize, int padding, int packMethod);
void UnloadFontData(CharInfo *chars, int charsCount);
void UnloadFont(Font font);

// Get the default Font
// Load font from file into GPU memory (VRAM)
// Load font from file with extended parameters
// Load font from Image (XNA style)
// Load font from memory buffer
// Load font data for further use
// Generate image font atlas using chars info
// Unload font chars info data (RAM)
// Unload Font from GPU memory (VRAM)

// Text drawing functions
void DrawFPS(int posX, int posY);
void DrawText(const char *text, int posX, int posY, int fontSize, Color color);
void DrawTextEx(Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint);
void DrawTextRec(Font font, const char *text, Rectangle rec, float fontSize, float spacing, bool wordWrap, Color tint);
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);
void DrawTextCodepoint(Font font, int codepoint, Vector2 position, float fontSize, Color tint);

// Shows current FPS
// Draw text (using default font)
// Draw text using font and additional parameters
// Draw text using font inside rectangle limits
// Draw text using font inside rectangle limits with support for text selection
// Draw one character (codepoint)

// Text misc. functions
int MeasureText(const char *text, int fontSize);
Vector2 MeasureTextEx(Font font, const char *text, float fontSize, float spacing);
int GetGlyphIndex(Font font, int codepoint);

// Measure string width for default font
// Measure string size for Font
// 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);
bool TextIsEqual(const char *text1, const char *text2);
unsigned int TextLength(const char *text);
const char *TextFormat(const char *text, ...);

// Copy one string to another, returns bytes copied
// Check if two text string are equal
// Get text length, checks for '\0' ending
// Text formatting with variables (sprintf style)

```

```
const char *TextSubtext(const char *text, int position, int length);
char *TextReplace(char *text, const char *replace, const char *by);
char *TextInsert(const char *text, const char *insert, int position);
const char *TextJoin(const char **textList, int count, const char *delimiter);
const char **TextSplit(const char *text, char delimiter, int *count);
void TextAppend(char *text, const char *append, int *position);
int TextFindIndex(const char *text, const char *find);
const char *TextToUpper(const char *text);
const char *TextToLower(const char *text);
const char *TextToPascal(const char *text);
int TextToInteger(const char *text);
char *TextToUtf8(int *codepoints, int length);

// UTF8 text strings management functions
int *GetCodepoints(const char *text, int *count);
int GetCodepointsCount(const char *text);
int GetNextCodepoint(const char *text, int *bytesProcessed);
const char *CodepointToUtf8(int codepoint, int *byteLength);

// Get a piece of a text string
// Replace text string (memory must be freed!)
// Insert text in a position (memory must be freed!)
// Join text strings with delimiter
// Split text into multiple strings
// Append text at specific position and move cursor!
// Find first text occurrence within a string
// Get upper case version of provided string
// Get lower case version of provided string
// Get Pascal case notation version of provided string
// Get integer value from text (negative values not supported)
// Encode text codepoint into utf8 text (memory must be freed!)

// Get all codepoints in a string, codepoints count returned by parameters
// Get total number of characters (codepoints) in a UTF8 encoded string
// Returns next codepoint in a UTF8 encoded string; 0x3f('?') is returned on
// 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);
void DrawPoint3D(Vector3 position, Color color);
void DrawCircle3D(Vector3 center, float radius, Vector3 rotationAxis, float rotationAngle, Color color);
void DrawTriangle3D(Vector3 v1, Vector3 v2, Vector3 v3, Color color);
void DrawTriangleStrip3D(Vector3 *points, int pointsCount, 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 DrawCubeTexture(Texture2D texture, Vector3 position, float width, float height, float length, 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);
void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color);
void DrawPlane(Vector3 centerPos, Vector2 size, Color color);
void DrawRay(Ray ray, Color color);
void DrawGrid(int slices, float spacing);
void DrawGizmo(Vector3 position);

// Draw a line in 3D world space
// Draw a point in 3D space, actually a small line
// Draw a circle in 3D world space
// Draw a color-filled triangle (vertex in counter-clockwise order!)
// Draw a triangle strip defined by points
// Draw cube
// Draw cube (Vector version)
// Draw cube wires
// Draw cube wires (Vector version)
// Draw cube textured
// Draw sphere
// Draw sphere with extended parameters
// Draw sphere wires
// Draw a cylinder/cone
// Draw a cylinder/cone wires
// Draw a plane XZ
// Draw a ray line
// Draw a grid (centered at (0, 0, 0))
// Draw simple gizmo

// Model loading/unloading functions
Model LoadModel(const char *fileName);
Model LoadModelFromMesh(Mesh mesh);
void UnloadModel(Model model);
void UnloadModelKeepMeshes(Model model);

// Load model from files (meshes and materials)
// Load model from generated mesh (default material)
// Unload model (including meshes) from memory (RAM and/or VRAM)
// Unload model (but not meshes) from memory (RAM and/or VRAM)

// Mesh loading/unloading functions
Mesh *LoadMeshes(const char *fileName, int *meshCount);
void UnloadMesh(Mesh mesh);
bool ExportMesh(Mesh mesh, const char *fileName);

// Load meshes from model file
// Unload mesh from memory (RAM and/or VRAM)
// Export mesh data to file, returns true on success

// Material loading/unloading functions
Material *LoadMaterials(const char *fileName, int *materialCount);
Material LoadMaterialDefault(void);
void UnloadMaterial(Material material);
void SetMaterialTexture(Material *material, int mapType, Texture2D texture);
void SetModelMeshMaterial(Model *model, int meshId, int materialId);

// Load materials from model file
// Load default material (Supports: DIFFUSE, SPECULAR, NORMAL maps)
// Unload material from GPU memory (VRAM)
// Set texture for a material map type (MAP_DIFFUSE, MAP_SPECULAR...)
// Set material for a mesh

// Model animations loading/unloading functions
ModelAnimation *LoadModelAnimations(const char *fileName, int *animsCount);
void UpdateModelAnimation(Model model, ModelAnimation anim, int frame);
void UnloadModelAnimation(ModelAnimation anim);
bool IsModelAnimationValid(Model model, ModelAnimation anim);

// Load model animations from file
// Update model animation pose
// Unload animation data
// Check model animation skeleton match

// Mesh generation functions
Mesh GenMeshPoly(int sides, float radius);
Mesh GenMeshPlane(float width, float length, int resX, int resZ);
Mesh GenMeshCube(float width, float height, float length);
Mesh GenMeshSphere(float radius, int rings, int slices);
Mesh GenMeshHemiSphere(float radius, int rings, int slices);
Mesh GenMeshCylinder(float radius, float height, int slices);
Mesh GenMeshTorus(float radius, float size, int radSeg, int sides);
Mesh GenMeshKnot(float radius, float size, int radSeg, int sides);
Mesh GenMeshHeightmap(Image heightmap, Vector3 size);
Mesh GenMeshCubicmap(Image cubicmap, Vector3 cubeSize);

// Generate polygonal mesh
// Generate plane mesh (with subdivisions)
// Generate cuboid mesh
// Generate sphere mesh (standard sphere)
// Generate half-sphere mesh (no bottom cap)
// Generate cylinder mesh
// Generate torus mesh
// Generate trefoil knot mesh
// Generate heightmap mesh from image data
// Generate cubes-based map mesh from image data

// Mesh manipulation functions
BoundingBox MeshBoundingBox(Mesh mesh);
void MeshTangents(Mesh *mesh);
void MeshBinormals(Mesh *mesh);
void MeshNormalsSmooth(Mesh *mesh);

// Compute mesh bounding box limits
// Compute mesh tangents
// Compute mesh binormals
// Smooth (average) vertex normals

// Model drawing functions
void DrawModel(Model model, Vector3 position, float scale, Color tint);
void DrawModelEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint);
void DrawModelWires(Model model, Vector3 position, float scale, Color tint);
void DrawModelWiresEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint);
void DrawBoundingBox(BoundingBox box, Color color);
void DrawBillboard(Camera camera, Texture2D texture, Vector3 center, float size, Color tint);
void DrawBillboardRec(Camera camera, Texture2D texture, Rectangle source, Vector3 center, float size, Color tint);

// Draw a model (with texture if set)
// Draw a model with extended parameters
// Draw a model wires (with texture if set)
// Draw a model wires (with texture if set) with ex
// Draw bounding box (wires)
// Draw a billboard texture
// Draw a billboard texture defined by source

// Collision detection functions
bool CheckCollisionSpheres(Vector3 center1, float radius1, Vector3 center2, float radius2);
bool CheckCollisionBoxes(BoundingBox box1, BoundingBox box2);
bool CheckCollisionBoxSphere(BoundingBox box, Vector3 center, float radius);
bool CheckCollisionRaySphere(Ray ray, Vector3 center, float radius);
bool CheckCollisionRaySphereEx(Ray ray, Vector3 center, float radius, Vector3 *collisionPoint);
bool CheckCollisionRayBox(Ray ray, BoundingBox box);
RayHitInfo GetCollisionRayMesh(Ray ray, Mesh mesh, Matrix transform);
RayHitInfo GetCollisionRayModel(Ray ray, Model model);
RayHitInfo GetCollisionRayTriangle(Ray ray, Vector3 p1, Vector3 p2, Vector3 p3);
RayHitInfo GetCollisionRayGround(Ray ray, float groundHeight);

// Detect collision between two spheres
// Detect collision between two bounding boxes
// Detect collision between box and sphere
// Detect collision between ray and sphere
// Detect collision between ray and sphere, returns collision point
// Detect collision between ray and box
// Get collision info between ray and mesh
// Get collision info between ray and model
// Get collision info between ray and triangle
// Get collision info between ray and ground plane (Y-normal plane)
```

## module: shaders (rLgL)

```
// Shader loading/unloading functions
Shader LoadShader(const char *vsFileName, const char *fsFileName);
Shader LoadShaderCode(const char *vsCode, const char *fsCode);
void UnloadShader(Shader shader);

// Load shader from files and bind default locations
// Load shader from code strings and bind default locations
// Unload shader from GPU memory (VRAM)

Shader GetShaderDefault(void);
Texture2D GetTextureDefault(void);
Texture2D GetShapesTexture(void);
Rectangle GetShapesTextureRec(void);
void SetShapesTexture(Texture2D texture, Rectangle source);

// Get default shader
// Get default texture
// Get texture to draw shapes
// Get texture rectangle to draw shapes
// Define default texture used to draw shapes

// Shader configuration functions
int GetShaderLocation(Shader shader, const char *uniformName);
int GetShaderLocationAttrib(Shader shader, const char *attribName);
void SetShaderValue(Shader shader, int uniformLoc, const void *value, int uniformType);
void SetShaderValueV(Shader shader, int uniformLoc, const void *value, int uniformType, int count);
void SetShaderValueMatrix(Shader shader, int uniformLoc, Matrix mat);
void SetShaderValueTexture(Shader shader, int uniformLoc, Texture2D texture);
void SetMatrixProjection(Matrix proj);
void SetMatrixModelview(Matrix view);
Matrix GetMatrixModelview(void);
Matrix GetMatrixProjection(void);

// Get shader uniform location
// Get shader attribute location
// Set shader uniform value
// Set shader uniform value vector
// Set shader uniform value (matrix 4x4)
// Set shader uniform value for texture
// Set a custom projection matrix (replaces internal projection matrix)
// Set a custom modelview matrix (replaces internal modelview matrix)
// Get internal modelview matrix
// Get internal projection matrix
```



```
// Shading begin/end functions
void BeginShaderMode(Shader shader);
void EndShaderMode(void);
void BeginBlendMode(int mode);
void EndBlendMode(void);

// VR control functions
void InitVrSimulator(void);
void CloseVrSimulator(void);
void UpdateVrTracking(Camera *camera);
void SetVrConfiguration(VrDeviceInfo info, Shader distortion);
bool IsVrSimulatorReady(void);
void ToggleVrMode(void);
void BeginVrDrawing(void);
void EndVrDrawing(void);

// Begin custom shader drawing
// End custom shader drawing (use default shader)
// Begin blending mode (alpha, additive, multiplied)
// End blending mode (reset to default: alpha blending)

// Init VR simulator for selected device parameters
// Close VR simulator for current device
// Update VR tracking (position and orientation) and camera
// Set stereo rendering configuration parameters
// Detect if VR simulator is ready
// Enable/Disable VR experience
// Begin VR simulator stereo rendering
// End VR simulator stereo rendering
```

## module: audio

```
// Audio device management functions
void InitAudioDevice(void);
void CloseAudioDevice(void);
bool IsAudioDeviceReady(void);
void SetMasterVolume(float volume);

// Wave/Sound loading/unloading functions
Wave LoadWave(const char *fileName);
Wave LoadWaveFromMemory(const char *fileType, const unsigned char *fileData, int dataSize); // Load wave from memory buffer
Sound LoadSound(const char *fileName);
Sound LoadSoundFromWave(Wave wave);
void UpdateSound(Sound sound, const void *data, int samplesCount);
void UnloadWave(Wave wave);
void UnloadSound(Sound sound);
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);
void PlaySoundMulti(Sound sound);
void StopSoundMulti(void);
int GetSoundsPlaying(void);
bool IsSoundPlaying(Sound sound);
void SetSoundVolume(Sound sound, float volume);
void SetSoundPitch(Sound sound, float pitch);
void WaveFormat(Wave *wave, int sampleRate, int sampleSize, int channels);
Wave WaveCopy(Wave wave);
void WaveCrop(Wave *wave, int initSample, int finalSample);
float *LoadWaveSamples(Wave wave);
void UnloadWaveSamples(float *samples);

// Music management functions
Music LoadMusicStream(const char *fileName);
void UnloadMusicStream(Music music);
void PlayMusicStream(Music music);
void UpdateMusicStream(Music music);
void StopMusicStream(Music music);
void PauseMusicStream(Music music);
void ResumeMusicStream(Music music);
bool IsMusicPlaying(Music music);
void SetMusicVolume(Music music, float volume);
void SetMusicPitch(Music music, float pitch);
float GetMusicTimeLength(Music music);
float GetMusicTimePlayed(Music music);

// 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);
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 SetAudioStreamBufferSizeDefault(int size);

// Initialize audio device and context
// Close the audio device and context
// Check if audio device has been initialized successfully
// Set master volume (listener)

// Load wave data from file
// Load wave from memory buffer
// Load sound from file
// Load sound from wave data
// Update sound buffer with new data
// Unload wave data
// Unload sound
// Export wave data to file, returns true on success
// Export wave sample data to code (.h), returns true on success

// Play a sound
// Stop playing a sound
// Pause a sound
// Resume a paused sound
// Play a sound (using multichannel buffer pool)
// Stop any sound playing (using multichannel buffer pool)
// Get number of sounds playing in the multichannel
// Check if a sound is currently playing
// Set volume for a sound (1.0 is max level)
// Set pitch for a sound (1.0 is base level)
// Convert wave data to desired format
// Copy a wave to a new wave
// Crop a wave to defined samples range
// Load samples data from wave as a floats array
// Unload samples data loaded with LoadWaveSamples()

// Load music stream from file
// Unload music stream
// Start music playing
// Updates buffers for music streaming
// Stop music playing
// Pause music playing
// Resume playing paused music
// Check if music is playing
// Set volume for music (1.0 is max level)
// Set pitch for a music (1.0 is base level)
// Get music time length (in seconds)
// Get current music time played (in seconds)

// Set volume for audio stream (1.0 is max level)
// Set pitch for audio stream (1.0 is base level)
// Default size for new audio streams
```

## 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
```