```
A simple and easy-to-use library to enjoy videogames programming
                                                                                              [raylib Discord server][github.com/raysan5/raylib]
                        raylib
                                                                                                 v3.5 quick reference card (download as PDF)
  module: core
                    // Window-related functions
void InitWindow(int width, int height, const char *title);
bool WindowShouldClose(void);
void CloseWindow(void);
bool IsWindowPeady(void);
bool IsWindowHulscreen(void);
bool IsWindowHulscreen(void);
bool IsWindowHaiminzed(void);
bool IsWindowHaiminzed(void);
bool IsWindowHaiminzed(void);
bool IsWindowState(unsigned int flag);
void SetWindowState(unsigned int flags);
void SetWindowState(unsigned int flags);
void ToggleFullscreen(void);
void MaximizeWindow(void);
void MaximizeWindow(void);
void SetWindowState(unsigned int flags);
void SetWindowFoit(Indey in in interplay);
void SetWindowFoit(Indey in interplay);
void SetWindowFoit(Indey in interplay);
void SetWindowFoit(int width, int height);
void SetWindowHoit(void);
int GetScreenHeight(void);
int GetScreenHeight(void);
int GetMonitorPosition(int monitor);
int GetMonitorPhysicalWidth(int monitor);
votor2 GetWindowScaleDPI(void);
vonst char *SetMonitorName(int monitor);
void SetClipboardText(const char *text);
const char *GetClipboardText(void);

// Cursor-related functions
void ShowCursor(void);
                                                                                                                                                                                                                                                                                                                                                                                                                          // Initialize window and OpenGL context
// Check if KEY_ESCAPE pressed or Close icon pressed
// Close window and unload OpenGL context
// Check if window has been initialized successfully
// Check if window has been initialized successfully
// Check if window is currently fullscreen
// Check if window is currently minimized (only PLATFORM_DESKTOP)
// Check if window is currently maximized (only PLATFORM_DESKTOP)
// Check if window is currently minimized (only PLATFORM_DESKTOP)
// Check if window is currently focused (only PLATFORM_DESKTOP)
// Check if window has been resized lost frome
// Check if window has been resized lost frome
// Check if window store successful sendbled
// Set window configuration state stags
// Clear window configuration state stags
// Clear window configuration state flags
// Clear window state: maximized, if resizable (only PLATFORM_DESKTOP)
// Set window state: moximized, if resizable (only PLATFORM_DESKTOP)
// Set window state: moximized, if resizable (only PLATFORM_DESKTOP)
// Set indow state: not minimized/maximized (only PLATFORM_DESKTOP)
// Set itle for window (only PLATFORM_DESKTOP)
// Set itle for window (only PLATFORM_DESKTOP)
// Set window position on screen (only PLATFORM_DESKTOP)
// Set window minimum dimensions (for FLAG_WINDOW_RESIZABLE)
// Set window dimensions
// Get current screen width
// Get current screen width
// Get specified monitor position
// Get specified monitor physical width in millimetres
// Get specified monitor physical height in millimetres
// Get specified monitor refresh rate
// Get specified monitor refresh rate
// Get specified monitor physical width in millimetres
// Get specified monitor physical width in millimetres
// Get specified monitor refresh rate
// Get specified monitor physical width in millimetres
// Get specified monitor refresh rate
// Get window scale DPI factor
// Get
                        // Cursor-related functions
void ShowCursor(void);
void HideCursor(void);
bool IsCursorHidden(void);
void EnableCursor(void);
void DisableCursor(void);
bool IsCursorOnScreen(void);
                                                                                                                                                                                                                                                                                                                                                                                                                            // Shows cursor
// Hides cursor
// Check if cursor is not visible
// Enables cursor (unlock cursor)
// Disables cursor (lock cursor)
// Check if cursor is on the current screen.
                           // Drawing-related functions
void ClearBackground(Color color);
void BeginDrawing(void);
void EndDrawing(void);
void BeginMode2D(Camera2D camera);
                                                                                                                                                                                                                                                                                                                                                                                                                            // Set background color (framebuffer clear color)
// Setup canvas (framebuffer) to start drawing
// End canvas drawing and swap buffers (double buffering)
// Initialize 2D mode with custom camera (2D)
// Ends 2D mode with custom camera (3D)
// Ends 3D mode and returns to default 2D orthographic mode
// Initializes render texture for drawing
// Ends drawing to render texture
// Begin scissor mode (define screen area for following drawing)
// End scissor mode
                           void EndMode2D(vaid);
void BeginNode3D(Camera3D camera);
void BeginNode3D(Camera3D camera);
void EndMode3D(void);
void BeginTextureMode(RenderTexture2D target);
void BeginTextureMode(void);
void BeginScissorMode(int x, int y, int width, int height);
void BeginScissorMode(void);
                        // Screen-space-related functions
Ray GetMouseRay(Vector2 mousePosition, Camera camera); // Returns a ray trace from mouse position
Matrix GetCameraMatrix(Camera camera); // Returns camera transform motrix (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);
int GetFPS(void);
float GetFrameTime(void);
double GetTime(void);
                                                                                                                                                                                                                                                                                                                                                                                                                              // Set target FPS (maximum)
// Returns current FPS
// Returns time in seconds for last frame drawn
// Returns elapsed time in seconds since InitWindow()
                           void SetConfigFlags(unsigned int flags);
                                                                                                                                                                                                                                                                                                                                                                                                                              // Setup init configuration flags (view FLAGS)
                           void SetTraceLogLevel(int logType);
void SetTraceLogExit(int logType);
void SetTraceLogCallback(TraceLogCallback callback);
void TraceLogCint logType, const char *text, ...);
                                                                                                                                                                                                                                                                                                                                                                                                                               // Set the current threshold (minimum) log level
// Set the exit threshold (minimum) log level
// Set a trace log callback to enable custom logging
// Show trace log messages (LOG_DEBUG, LOG_INFO, LOG_WARNING, LOG_ERROR)
                           void *MemAlloc(int size);
void MemFree(void *ptr);
void TakeScreenshot(const char *fileName);
int GetRandomValue(int min, int max);
                                                                                                                                                                                                                                                                                                                                                                                                                              // Internal memory allocator
// Internal memory free
// Takes a screenshot of current screen (saved a .png)
// Returns a random value between min and max (both included)
                     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);
int LoadStorageValue(unsigned int position);
                                                                                                                                                                                                                                                                                                                                                                                                                            // Save integer value to storage file (to defined position), returns true on success // 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: keyb
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);
                                                                                                                                                                                                                                                                                                                                                                                                                              // 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 key ressed (unicode), call it multiple times for chars queued
                        // 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 IsGamepadButtonPressed(void);
int GetGamepadAxisCount(int gamepad);
float GetGamepadAxisMovement(int gamepad, int axis);
                                                                                                                                                                                                                                                                                                                                                                                                                            // 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
// Bet the last gamepad button pressed
// Return gamepad axis count for a gamepad
// Return axis movement value for a gamepad 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 SetMouseOffset(int offsetX, int offsetY);
int GetMouseWheelMove(void);
int GetMouseWheelMove(void);
void SetMouseCursor(void);
void SetMouseCursor(void);
                                                                                                                                                                                                                                                                                                                                                                                                                           // 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 position XY
// Set mouse soffset
// Set mouse scaling
// Returns mouse wheel movement Y
// Returns mouse cursor if (MouseCursor enum)
// Set mouse cursor
                            // Input-related functions: touch
int GetTouchX(void);
int GetTouchY(void);
Vector2 GetTouchPosition(int index);
                                                                                                                                                                                                                                                                                                                                                                                                                               // 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 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 nable delta
                                                                                                                                                                                                                                                                                                                                                                                                                               // 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
                        // 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
                     // Basis shows drowing functions
void DemaPixel(Vicetor2 position, Color color);
void DemaPixel(Vicetor2 startPos, Vector2 endPos, Color color);
void DemaLine(Vicetor2 startPos, Vector2 endPos, Color color);
void DemaLine(Vicetor2 startPos, Vector2 endPos, Color color);
void DemaLine(Vicetor2 startPos, Vector2 endPos, Float thick, Color color);
void DemaLine(Vicetor2 position, to printsCourt, Color color);
void DemaCircle(int centerX, int centerY, float radius, int startAngle, int endAngle, int segments, Color color);
void DemaCircleSector(Vicetor2 center, float radius, int startAngle, int endAngle, int segments, Color color);
void DemaCircleSector(Vicetor2 center, float radius, Color color);
void DemaCircleSector(vicetor2 center, float radius, Color color);
void DemaCircleSector(vicetor2 center, float radius, Color color);
void DemaCircle(Vicetor2 center, float radius, float radius, Color color);
void DemaCircle(vicetor2 center, float radius, float radius,
                        module: textures
                         // Image loading functions
// NOTE: This functions do not require GPU access
Image LoadImage(Const char *fileName);
Image LoadImage(Rost char *fileName, int width, int height, int format, int headerSize);
Image LoadImageAnim(const char *fileName, int *frames);
Image LoadImage(TomMemory(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 su
                         // Image generation functions
Image GenImageColor(int width, int height, Color 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 col1, Color col2);
Image GenImagePerLinkoise(int width, int height, float factor);
Image GenImagePerLinkoise(int width, int height, int offsetX, int offsetY, float scale);
Image GenImageCellular(int width, int height, int tileSize);
                     // Image drawing functions
// NOTE: Image software-rendering functions (CPU)
void ImageClearBackground(Image *dst, Color color);
// Clear image background with given color
void ImageClearPackground(Image *dst, int posX, int posY, Color color);
// Draw pixel within an image
void ImageClorewPixel(Image *dst, Vector2 position, Color color);
// Draw pixel within an image (Vector version)
void ImageClorewLineV(Image *dst, Vector2 start, Vector2 end, Color color);
// Draw line within an image (Vector version)
void ImageClorewCircle(Image *dst, int centerX, int centerY, int radius, Color color);
// Draw circle within an image (Vector version)
void ImageClorewCircle(Image *dst, Vector2 center, int radius, Color color);
// Draw circle within an image (Vector version)
void ImageClorewCircleV(Image *dst, int posX, int posX, int width, int height, Color color);
// Draw circle within an image (Vector version)
void ImageClorewRectangleV(Image *dst, Vector2 position, Vector2 size, Color color);
// Draw rectangle within an image (Vector version)
void ImageClorewRectangleLines(Image *dst, Rectangle rec, Color color);
// Draw rectangle within an image (Vector version)
void ImageClorewRectangleLines(Image *dst, Rectangle rec, int thick, Color color);
// Draw rectangle within an image (vector version)
void ImageDrawText(Image *dst, Image src, Rectangle srcRec, Rectangle dstRec, Color tint);
// Draw a source image within an destination image (tint applied to source)
void ImageDrawText(Image *dst, Const char *text, int posX, int posX, int fontSize, Color color);
// Draw text (using defoult font) within an image (destination)
void ImageDrawTextEx(Image *dst, Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint); // Draw text (custom sprite font) within an image
                         // Texture loading functions
// NOTE: These functions require GPU access
Texture2D LoadTexture(const char *fiteName);
Texture2D LoadTextureFromImage(Image image);
TextureCubemap LoadTextureCubemap(Image image, int layoutType);
RenderTexture2D LoadRenderTexture(int width, int height);
void UnloadTexture(Texture2D texture);
void UnloadTexture(Texture2D texture);
void UpdateTexture(Texture2D texture, const void *pixels);
void UpdateTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderTextureRenderT
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             // Load texture from file into GPU memory (VRAM)
// Load texture from image data
// Load cubemop from image, multiple image cubemap layouts supported
// Load texture from GPU memory (VRAM)
// 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)
                           // Texture configuration functions
void GenTextureMipmaps(Texture2D *texture);
void SetTextureFither(Texture2D texture, int filterMode);
void SetTextureWrap(Texture2D texture, int wrapMode);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                // Generate GPU mipmaps for a texture
// Set texture scaling filter mode
// Set texture wrapping mode
                        // Texture drawing functions
void DrawTexture(Texture2D texture, int posX, int posY, Color tint);
void DrawTexture(Texture2D texture, vector2 position, Color tint);
void DrawTexture(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 DrawTextureRe(Texture2D texture, Rectangle source, Vector2 position, Color tint);
// Draw a Texture2D with extended parameters
void DrawTextureQuad(Texture2D texture, Vector2 position, Color tint);
// Draw texture quad with tiling and offset parameters
void DrawTexturePlate(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, float scale, Color tint);
// Draw part of a texture (defined by a rectangle source)
void DrawTexturePlate(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, Color tint);
// Draw a part of a texture defined by a rectangle dest, Vector2 origin, float rotation, Color tint);
// Draw a part of a texture defined by a rectangle dest, Vector2 origin, float rotation, Color tint);
// Draw a part of a texture defined by a rectangle dest, Vector2 origin, float rotation, Color tint);
// Draw a part of a texture defined by a rectangle dest, Vector2 origin, float rotation, Color tint);
// Draw a part of a texture defined by a rectangle dest, Vector2 origin, float rotation, Color tint);
// Draw a texture (or part of it) that stretch.
                        // Color/pixel related functions
Color Fade(Color color, float alpha);
int ColorToInt(Color color);
Vector4 ColorNormalize(Color color);
Color ColorFromNormalized(Vector4 normalized);
Vector3 ColorToNSV(Color color);
Color ColorFromNSV(Float hue, float saturation, float value);
Color ColorAlpha(Color color, float alpha);
Color ColorAlpha(Eolor color, float alpha);
Color ColorAlpha(Eolor color, float saturation, float value);
Color ColorAlpha(Eolor color, float alpha);
Color ColorAlpha(Eolor color, float alpha);
Color ColorAlpha(Eolor color, float alpha);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              // 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 KSV values for a Color
// Returns HSV values for a Color
// Returns a Color from HSV values
// Returns a Color from HSV values
// Returns soc 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
                         Color GetColor(int hexValue);
Color GetPixelColor(void *srcPtr, int format);
void SetPixelColor(void *srcPtr, Color color, int format);
int GetPixelDataSize(int width, int height, int format);
 module: text
                        // Font loading/unloading functions
Font GetFontDefault(void);
Font LoadFont(const char *fileName);
Font LoadFont(const char *fileName);
Font LoadFontEx(const char *fileName);
Font LoadFontEx(const char *fileName, int fontSize, int *fontChars, int charsCount);
Font LoadFontFromImage(Image image, Color key, int firstChar);
Font LoadFontFromMage(Image image, Color key, int firstChar);
Font LoadFontFromMemory(const char *fileType, const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount);
Font LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount, int type);
Font LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount, int type);
Font LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount, int type);
Font LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount, int charsCount, int type);
Font LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int charsCount, int charsCount, int type);
Font LoadFontData(charInfo *chars, kectangle **recs, int charsCount, int fontSize, int *fontChars, int charsCount, int type);
Font LoadFontData(CharInfo *chars, int charsCount);
Font Load font from GPU memory (VRAM)

### VICTUAL COUNTY OF THE COU
                        // Text drawing functions
void DrawFPS(int posX, int posY);
void DrawFeX(int posX, int posY, int fontSize, Color color);
// Shows current FPS
void DrawTextEx(Font font, 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 DrawTextRecEx(Font font, const char *text, Rectangle rec, float spacing, bool wordWrap, Color tint);
// Draw text using font inside rectangle limits
int selectStart, int selectLength, Color selectTint, Color selectBackTint);
// Draw text using font inside rectangle limits with support for text selective void DrawTextCodepoint(Font font, int codepoint, Vector2 position, float fontSize, Color tint);
// 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
```

```
// Basic geometric 3D shapes drawing functions

void DrawLine3D(Vector3 startPos, Vector3 endPos, Color color);

void DrawCincte3D(Vector3 position, Color color);

void DrawCincte3D(Vector3 position, Color color);

void DrawCincte3D(Vector3 val, Vector3 v2, Vector3 v3, Color color);

void DrawTriangle3D(Vector3 v2, Vector3 v3, Color color);

void DrawTriangle3D(Vector3 v2, Vector3 v3, Color color);

void DrawTriangle3D(Vector3 position, float width, float height, float length, Color color);

void DrawCube(Vector3 position, Vector3 size, Color color);

void DrawCubeVevector3 position, Vector3 size, Color color);

void DrawCubeWires(Vector3 position, Vector3 size, Color color);

void DrawCubeWires(Vector3 position, Vector3 size, Color color);

void DrawSubeVevector3 position, Vector3 size, Color color);

void DrawSubeVevector3 centerPos, float radius, Color color);

void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color color);

void DrawSphereWires(Vector3 position, float radius float radiusBottom, float height, int slices, Color color);

void DrawSphereWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color);

void DrawSphereWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color);

void DrawSphereWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color);

void DrawSqu(rad radius float radiusTop, float radiusBottom, float height, int slices, Color color);

void DrawSqu(rad radius float spacing);

void DrawSqu(rad radiu
                        // Model loading/unloading functions
Model LoadModel(const char *ficklame);
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 GenMeshPolac(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 GenMeshHot(float radius, float size, int radSeg, int sides);
Mesh GenMeshHot(float radius, float size, int radSeg, int sides);
Mesh GenMeshHot(float radius, float size, int radSeg, int sides);
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 torus mesh
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              // Generate torus mesh
// Generate trefoil knot mesh
// Generate heightmap mesh from image data
// Generate cubes-based map mesh from image data
                          // Hesh monapotation functions
BoundingBox MeshBoundingBox(Mesh mesh);
void MeshTangents(Mesh *mesh);
void MeshBinormals(Mesh *mesh);
void MeshNormalsSmooth(Mesh *mesh);
                       // 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 DrawModelEx(Model model, Vector3 position, float scale, Color tint);

// Draw a model wires (with texture if set)

void DrawModelWires(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint);

void DrawModelWires(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint);

void DrawBoundingBox(BoundingBox box, Color color);

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 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 CheckCollisionRayBox(Ray ray, BoundingBox box);
RayHitInfo GetCollisionRayHosh(Ray ray, Wesh mesh, Matrix transform);
RayHitInfo GetCollisionRayModel(Ray ray, Model model);
RayHitInfo GetCollisionRayTriangle(Ray ray, Vector3 p1, Vector3 p2, Vector3 p3);
RayHitInfo GetCollisionRayTriangle(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 mede
// Get collision info between ray and triangle
// Get collision info between ray and ground plane (Y-normal plane)
module: shaders (rtol)
                          // 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);
                        // Shader configuration functions
int GetShaderLocation(Shader shader, const char *uniformName);
int GetShaderLocation(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 SetShaderValueV(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);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    // 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)
                           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 BedinVrDrawing(void);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 // 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
// Enoble/Disable VR experience
// Begin VR simulator stereo rendering
// End VR simulator stereo rendering
```

// 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)
// 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 accurrence 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; 08x5f('?') is returned on // Encode codepoint into utf8 text (char array length returned as parameter)

int GetGlyphIndex(Font font, int codepoint);

// 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 *sec);
bool TextIsEqual(const char *text), const char *text2);
unsigned int TextLength(const char *text), const char *TextFormat(const char *text, int position, int length);
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, int sount, const char *delimiter);
const char *TextSplit(const char *text, tint count, const char *delimiter);
const char *TextSplit(const char *text, const char *find);
const char *TextToUpper(const char *text);
const char *TextToUpper(const char *text);
const char *TextToUpper(const char *text);
const char *TextToPascal(const char *text);
int TextToInteger(const char *text);

int *GetCodepoints(const char *text, int *count); int GetCodepointsCount(const char *text); int GetNatCodepoint(const char *text); int GetNatCodepoint(const char *text, int *bytesProcessed); const char *CodepointToUtf8(int codepoint, int *byteLength);

module: audio

```
void InitAudioDevice(void);
void CloseAudioDevice(void);
bool IsAudioDeviceReady(void);
void SetMasterVolume(float volume);
                                                                                                                                                                                                                                                                                                                                                                                                           // Initialize audio device and context
// Close the audio device and context
// Check if audio device has been initialized successfully
// Set master volume (listener)
// Wave/Sound loading/unloading functions

// Wave LoadWave (const char *fileName);

// Load wave data from file

// Load wave from memory buffer

// Load sound from file

// Load sound from file

// Load sound from wave data

// Unload sound sound, const void *data, int samplesCount);

// Update sound buffer with new data

// Unload wave data

// Unload sound

// Unload sound

// Unload sound

// Unload sound

// Export wave data to file, returns true on success

// Export wave sample data to code (.h), returns true on success

// Export wave sample data to code (.h), returns true on success
  void PlaySound(Sound sound);
                                                                                                                                                                                                                                                                                                                                                                                                           // Play a sound
// Stop playing a sound
  void StopSound(Sound sound)
     void PauseSound(Sound sound);
void ResumeSound(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 SetSoundVolume(Sound sound, float pitch);
void WaveFormat(Wave *wave, int sampleRate, int sampleSize, int channels);
Wave WaveCopy(Wave wave);
void WaveCorp(Wave *wave, int initSample, int finalSample);
float *LoadWaveSamples(Wave wave);
void UnloadWaveSamples(float *samples);
                                                                                                                                                                                                                                                                                                                                                                                                                     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 dato to desired format
Conu a wave to a new wave
                                                                                                                                                                                                                                                                                                                                                                                                            // 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)
 // Music management functions Music LoadMusicStream(Const char *fit void UnloadMusicStream(Music music); void UpdateMusicStream(Music music); void UpdateMusicStream(Music music); void StopMusicStream(Music music); void StopMusicStream(Music music);
                        PauseMusicStream(Music music);
ResumeMusicStream(Music music);
 World Messumenosite ream(note mostle),
bool IsMusicPlaying(Music music);
void SetMusicVolume(Music music, float volume);
void SetMusicPitch(Music music, float pitch);
float GetMusicTimePlayed(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); // Update audio stream buffers with data
void CloseAudioStream(AudioStream stream); // Close audio stream and free amemory
bool IsAudioStreamProcessed(AudioStream stream); // Check if any audio stream buffers requires refill
                        PlayAudioStream(AudioStream stream);
                                                                                                                                                                                                                                                                                                                                                                                                            // Play audio stream
// Pause audio stream
// Resume audio stream
// Check if audio stream is playing
// Stop audio stream
// 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
  void PlayAudioStream(AudioStream stream);
void PauseAudioStream(AudioStream stream);
void ResumeAudioStream(AudioStream stream);
bool IsAudioStreamPlaying(AudioStream stream);
void StopAudioStream(AudioStream stream);
void SetAudioStream(AudioStream stream, float volume);
void SetAudioStreamPlayIng(AudioStream stream, float pitch);
void SetAudioStreamPltch(AudioStream stream, float pitch);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          // Custom raylib color palette for amazing visuals #define LIGHTGRAY (Color){ 200, 200, 200, 255 } #define BRKGRAY (Color){ 303, 436, 130, 130, 255 } #define DARKGRAY (Color){ 88, 80, 80, 255 } #define GOLD (Color){ 255, 203, 0, 255 } #define GOLD (Color){ 255, 203, 0, 255 } #define RED (Color){ 255, 160, 194, 255 } #define RED (Color){ 230, 41, 55, 255 } #define RED (Color){ 230, 41, 55, 255 } #define BREEN (Color){ 190, 33, 55, 255 } #define DARKGREEN (Color){ 0, 228, 48, 255 } #define DARKGREEN (Color){ 0, 117, 44, 255 } #define BARKGREEN (Color){ 0, 117, 44, 255 } #define PURPLE (Color){ 102, 191, 255, 255 } #define PURPLE (Color){ 0, 82, 172, 255 } #define PURPLE (Color){ 1, 23, 60, 190, 255 } #define BROWN (Color){ 211, 176, 131, 255 } #define BROWN (Color){ 277, 166, 79, 255 } #define DARKBROWN (Color){ 255, 255, 255, 255, 255 } $ #define DARKBROWN (Color){ 255, 255, 255, 255, 255 } $ #define DARKBROWN (Color){ 255, 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255, 255 } $ #define WHITE (Color){ 255, 255, 255, 255, 255 } $ #define WHITE
                                                                                                                     // Vector2 type

// Vector3 type

// Vector4 type

// Quaternion type

// Mutrix type (OpenGL style)

// Color type, RGBA (32bit)

// Rectangle type
  struct Vector2;
struct Vector3;
struct Vector4;
struct Quaternion;
struct Matrix;
struct Color;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                // Light Gray
// Gray
// Dark Gray
// Yellow
// Gold
// Orange
// Pink
// Red
  struct Rectangle;
                                                                                                                      // Image type (multiple pixel formats supported)
// NOTE: Data stored in CPU memory (RAM)
// Texture type (multiple internal formats supported)
// NOTE: Data stored in GPU memory (VRAM)
// RenderTexture type, for texture rendering
// N-Patch layout info
// Font tharacter info
// Font type, includes texture and chars data
 struct Image;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   // Maroon
 struct Texture:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                // Green
// Lime
// Dark Green
// Sky Blue
// Blue
// Dark Blue
// Purple
// Violet
// Dark Purple
// Beige
// Brown
 struct RenderTexture;
struct NPatchInfo;
struct CharInfo;
struct Font;
                                                                                                                     // Camera type, defines 3d camera position/orientation
// Camera2D type, defines a 2d camera
// Vertex data definning a mesh
// Shader type (generic shader)
// Material texture map
// Material type
// Basic 3d Model type
// Transformation (used for bones)
// Bone information
// Model animation data (bones and frames)
// Ray type (useful for raycast)
// Raycast hit information
// Bounding box type for 3d mesh
  struct Camera;
  struct Camera2D;
  struct Mesh;
  struct Shader:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    // Brown
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   // Dark Brown
     struct MaterialMan:
   struct Material;
 struct Material;
struct Model;
struct Transform;
struct BoneInfo;
struct ModelAnimation;
struct Ray;
struct RayHitInfo;
struct BoundingBox;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              #define WHITE
#define BLACK
#define BLANK
#define MAGENTA
#define RAYWHITE
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        (Color){ 255, 255, 255, 255 }
(Color){ 0, 0, 0, 255 }
(Color){ 0, 0, 0, 0 }
(Color){ 255, 0, 255, 255 }
(Color){ 245, 245, 245, 255 }
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                // White
// Black
// Transparent
// Magenta
// Ray White
```