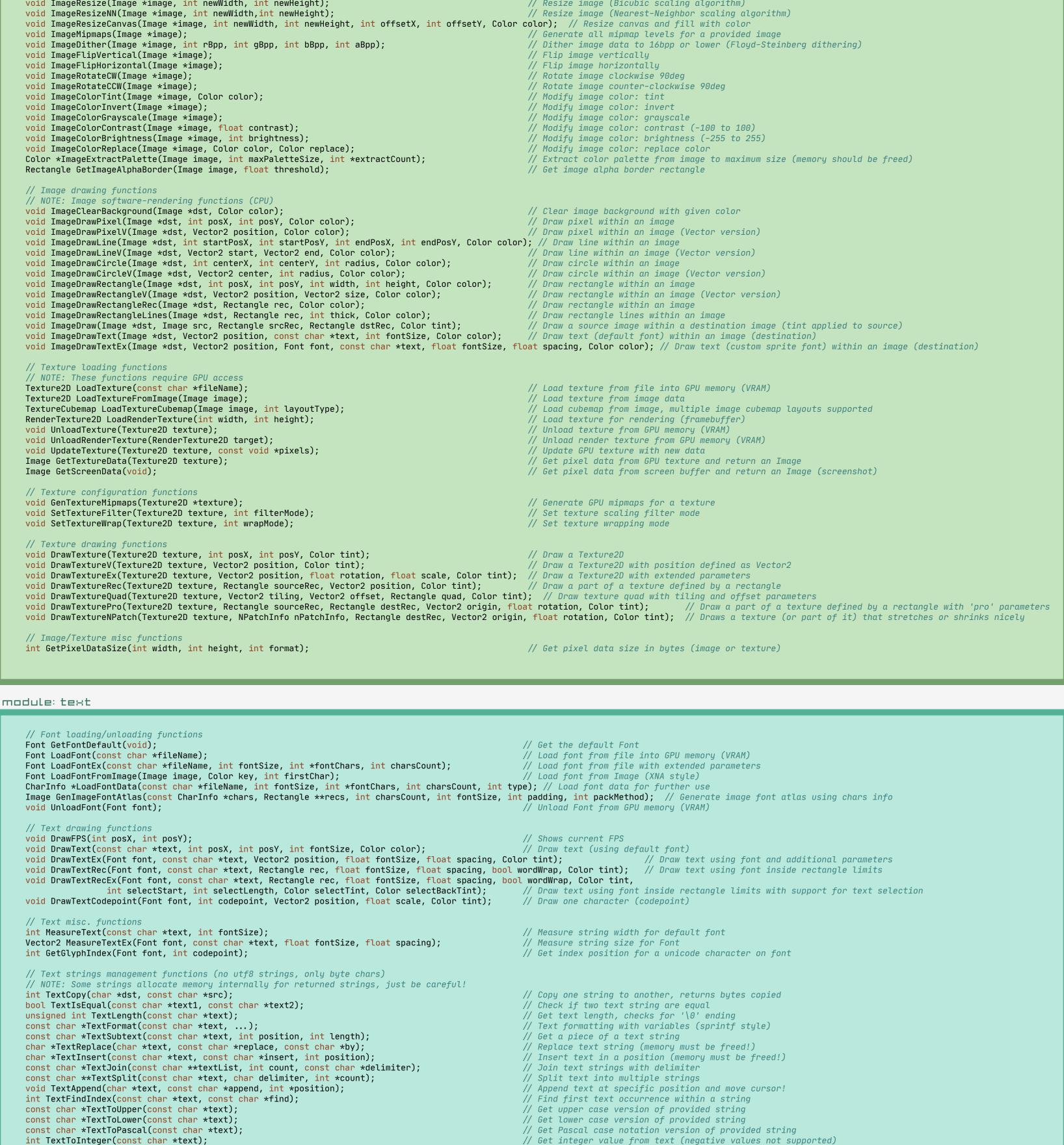
A simple and easy-to-use library to enjoy videogames programming [raylib Discord server][github.com/raysan5/raylib] raylib v3.0 quick reference card (download as PDF) module: core // Window-related functions void InitWindow(int width, int height, const char \*title); // Initialize window and OpenGL context bool WindowShouldClose(void); // Check if KEY\_ESCAPE pressed or Close icon pressed void CloseWindow(void); // Close window and unload OpenGL context bool IsWindowReady(void); // Check if window has been initialized successfully bool IsWindowMinimized(void); // Check if window has been minimized (or lost focus) bool IsWindowResized(void); // Check if window has been resized bool IsWindowHidden(void); // Check if window is currently hidden bool IsWindowFullscreen(void); // Check if window is currently fullscreen void ToggleFullscreen(void); // Toggle fullscreen mode (only PLATFORM\_DESKTOP) void UnhideWindow(void); // Show the window void HideWindow(void); // Hide the window void SetWindowIcon(Image image); // Set icon for window (only PLATFORM\_DESKTOP) void SetWindowTitle(const char \*title); // Set title for window (only PLATFORM\_DESKTOP) void SetWindowPosition(int x, int y); // Set window position on screen (only PLATFORM\_DESKTOP) void SetWindowMonitor(int monitor); // Set monitor for the current window (fullscreen mode) void SetWindowMinSize(int width, int height); // Set window minimum dimensions (for FLAG\_WINDOW\_RESIZABLE) void SetWindowSize(int width, int height); // Set window dimensions void \*GetWindowHandle(void); // Get native window handle int GetScreenWidth(void); // Get current screen width int GetScreenHeight(void); // Get current screen height int GetMonitorCount(void); // Get number of connected monitors int GetMonitorWidth(int monitor); // Get primary monitor width // Get primary monitor height int GetMonitorHeight(int monitor); int GetMonitorPhysicalWidth(int monitor); // Get primary monitor physical width in millimetres int GetMonitorPhysicalHeight(int monitor); // Get primary monitor physical height in millimetres Vector2 GetWindowPosition(void); // Get window position XY on monitor const char \*GetMonitorName(int monitor); // Get the human-readable, UTF-8 encoded name of the primary monitor const char \*GetClipboardText(void); // Get clipboard text content void SetClipboardText(const char \*text); // Set clipboard text content // Cursor-related functions void ShowCursor(void); // Shows cursor void HideCursor(void); // Hides cursor bool IsCursorHidden(void); // Check if cursor is not visible void EnableCursor(void); // Enables cursor (unlock cursor) void DisableCursor(void); // Disables cursor (lock cursor) // Drawing-related functions void ClearBackground(Color color); // Set background color (framebuffer clear color) void BeginDrawing(void); // Setup canvas (framebuffer) to start drawing void EndDrawing(void); // End canvas drawing and swap buffers (double buffering) void BeginMode2D(Camera2D camera); // Initialize 2D mode with custom camera (2D) void EndMode2D(void); // Ends 2D mode with custom camera void BeginMode3D(Camera3D camera); // Initializes 3D mode with custom camera (3D) void EndMode3D(void); // Ends 3D mode and returns to default 2D orthographic mode void BeginTextureMode(RenderTexture2D target); // Initializes render texture for drawing void EndTextureMode(void); // Ends drawing to render texture void BeginScissorMode(int x, int y, int width, int height); // Begin scissor mode (define screen area for following drawing) void EndScissorMode(void); // End scissor mode // Screen-space-related functions Ray GetMouseRay(Vector2 mousePosition, Camera camera); // Returns a ray trace from mouse position // Returns camera transform matrix (view matrix) Matrix GetCameraMatrix(Camera camera); 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 α 3d world space position Vector2 GetWorldToScreen2D(Vector2 position, Camera2D camera); // Returns the screen space position for a 2d camera world space position // Returns the world space position for a 2d camera screen space position Vector2 GetScreenToWorld2D(Vector2 position, Camera2D camera); // Timing-related functions void SetTargetFPS(int fps); // Set target FPS (maximum) int GetFPS(void); // Returns current FPS float GetFrameTime(void); // Returns time in seconds for last frame drawn double GetTime(void); // Returns elapsed time in seconds since InitWindow() // Color-related functions int ColorToInt(Color color); // Returns hexadecimal value for a Color Vector4 ColorNormalize(Color color); // Returns color normalized as float [0..1] // Returns color from normalized values [0..1] Color ColorFromNormalized(Vector4 normalized); Vector3 ColorToHSV(Color color); // Returns HSV values for a Color Color ColorFromHSV(Vector3 hsv); // Returns a Color from HSV values Color GetColor(int hexValue); // Returns a Color struct from hexadecimal value Color Fade(Color color, float alpha); // Color fade-in or fade-out, alpha goes from 0.0f to 1.0f // Misc. functions void SetConfigFlags(unsigned int flags); // Setup window configuration flags (view FLAGS) void SetTraceLogLevel(int logType); // Set the current threshold (minimum) log level void SetTraceLogExit(int logType); // Set the exit threshold (minimum) log level void SetTraceLogCallback(TraceLogCallback callback); // Set a trace log callback to enable custom logging void TraceLog(int logType, const char \*text, ...); // Show trace log messages (LOG\_DEBUG, LOG\_INFO, LOG\_WARNING, LOG\_ERROR) void TakeScreenshot(const char \*fileName); // Takes a screenshot of current screen (saved a .png) int GetRandomValue(int min, int max); // Returns a random value between min and max (both included) // Files management functions unsigned char \*LoadFileData(const char \*fileName, int \*bytesRead); // Load file data as byte array (read) void SaveFileData(const char \*fileName, void \*data, int bytesToWrite); // Save data to file from byte array (write) char \*LoadFileText(const char \*fileName); // Load text data from file (read), returns a '\0' terminated string void SaveFileText(const char \*fileName, char \*text); // Save text data to file (write), string must be '\0' terminated bool FileExists(const char \*fileName); // Check if file exists bool IsFileExtension(const char \*fileName, const char \*ext); // Check file extension bool DirectoryExists(const char \*dirPath); // Check if a directory path exists const char \*GetExtension(const char \*fileName); // Get pointer to extension for a filename string const char \*GetFileName(const char \*filePath); // Get pointer to filename for a path string const char \*GetFileNameWithoutExt(const char \*filePath); // Get filename string without extension (uses static string) const char \*GetDirectoryPath(const char \*filePath); // Get full path for a given fileName with path (uses static string) const char \*GetPrevDirectoryPath(const char \*dirPath); // Get previous directory path for a given path (uses static string) const char \*GetWorkingDirectory(void); // Get current working directory (uses static string) char \*\*GetDirectoryFiles(const char \*dirPath, int \*count); // Get filenames in a directory path (memory should be freed) void ClearDirectoryFiles(void); // Clear directory files paths buffers (free memory) bool ChangeDirectory(const char \*dir); // Change working directory, returns true if success bool IsFileDropped(void); // Check if a file has been dropped into window char \*\*GetDroppedFiles(int \*count); // Get dropped files names (memory should be freed) void ClearDroppedFiles(void); // Clear dropped files paths buffer (free memory) long GetFileModTime(const char \*fileName); // Get file modification time (last write time) unsigned char \*CompressData(unsigned char \*data, int dataLength, int \*compDataLength); // Compress dαtα (DEFLATE αlgorythm) unsigned char \*DecompressData(unsigned char \*compData, int compDataLength, int \*dataLength); // Decompress data (DEFLATE αlgorythm) // Persistent storage management int LoadStorageValue(int position); // Load integer value from storage file (from defined position) void SaveStorageValue(int position, int value); // Save integer value to storage file (to defined position) void OpenURL(const char \*url); // Open URL with default system browser (if available) //-----// Input Handling Functions // Input-related functions: keyb bool IsKeyPressed(int key); // Detect if a key has been pressed once bool IsKeyDown(int key); // Detect if a key is being pressed bool IsKeyReleased(int key); // Detect if a key has been released once bool IsKeyUp(int key); // Detect if a key is NOT being pressed int GetKeyPressed(void); // Get latest key pressed void SetExitKey(int key); // Set a custom key to exit program (default is ESC) // Input-related functions: gamepads bool IsGamepadAvailable(int gamepad); // Detect if a gamepad is available bool IsGamepadName(int gamepad, const char \*name); // Check gamepad name (if available) const char \*GetGamepadName(int gamepad); // Return gamepad internal name id bool IsGamepadButtonPressed(int gamepad, int button); // Detect if a gamepad button has been pressed once bool IsGamepadButtonDown(int gamepad, int button); // Detect if a gamepad button is being pressed bool IsGamepadButtonReleased(int gamepad, int button); // Detect if a gamepad button has been released once bool IsGamepadButtonUp(int gamepad, int button); // Detect if a gamepad button is NOT being pressed // Get the last gamepad button pressed int GetGamepadButtonPressed(void); int GetGamepadAxisCount(int gamepad); // Return gamepad axis count for a gamepad float GetGamepadAxisMovement(int gamepad, int axis); // Return axis movement value for a gamepad axis // Input-related functions: mouse bool IsMouseButtonPressed(int button); // Detect if a mouse button has been pressed once bool IsMouseButtonDown(int button); // Detect if a mouse button is being pressed bool IsMouseButtonReleased(int button); // Detect if a mouse button has been released once // Detect if a mouse button is NOT being pressed bool IsMouseButtonUp(int button); int GetMouseX(void); // Returns mouse position X int GetMouseY(void); // Returns mouse position Y Vector2 GetMousePosition(void); // Returns mouse position XY void SetMousePosition(int x, int y); // Set mouse position XY void SetMouseOffset(int offsetX, int offsetY); // Set mouse offset void SetMouseScale(float scaleX, float scaleY); // Set mouse scaling int GetMouseWheelMove(void); // Returns mouse wheel movement Y // Input-related functions: touch int GetTouchX(void); // Returns touch position X for touch point 0 (relative to screen size) int GetTouchY(void); // Returns touch position Y for touch point 0 (relative to screen size) Vector2 GetTouchPosition(int index); // Returns touch position XY for a touch point index (relative to screen size) // Gestures and Touch Handling Functions (Module: gestures) void SetGesturesEnabled(unsigned int gestureFlags);

// Enable α set of gestures using flags bool IsGestureDetected(int gesture); // Check if a gesture have been detected int GetGestureDetected(void); // Get latest detected gesture int GetTouchPointsCount(void); // Get touch points count float GetGestureHoldDuration(void); // Get gesture hold time in milliseconds Vector2 GetGestureDragVector(void); // Get gesture drag vector // Get gesture drag angle float GetGestureDragAngle(void); // Get gesture pinch delta Vector2 GetGesturePinchVector(void); // Get gesture pinch angle float GetGesturePinchAngle(void); void SetCameraMode(Camera camera, int mode);

// Set camera mode (multiple camera modes available)

// Undata camera position for selected mode void UpdateCamera(Camera \*camera); // Update camera position for selected mode void SetCameraPanControl(int panKey); // Set camera pan key to combine with mouse movement (free camera) void SetCameraAltControl(int altKey); // Set camera alt key to combine with mouse movement (free camera) void SetCameraSmoothZoomControl(int szKey); // Set camera smooth zoom key to combine with mouse (free camera) void SetCameraMoveControls(int frontKey, int backKey, int rightKey, int leftKey, int upKey, int downKey); // Set camera move controls (1st person and 3rd person cameras) module: shapes // Basic shapes drawing functions void DrawPixel(int posX, int posY, Color color); // Draw a pixel void DrawPixelV(Vector2 position, Color color); // Draw a pixel (Vector version) void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color); // Draw a line void DrawLineV(Vector2 startPos, Vector2 endPos, Color color); // Draw a line (Vector version) void DrawLineEx(Vector2 startPos, Vector2 endPos, float thick, Color color); // Draw a line defining thickness void DrawLineBezier(Vector2 startPos, Vector2 endPos, float thick, Color color); // Draw a line using cubic-bezier curves in-out // Draw lines sequence void DrawLineStrip(Vector2 \*points, int numPoints, Color color); void DrawCircle(int centerX, int centerY, float radius, Color color); // Draw a color-filled circle void DrawCircleSector(Vector2 center, float radius, int startAngle, int endAngle, int segments, Color color); // Draw α piece of α circle void DrawCircleSectorLines(Vector2 center, float radius, int startAngle, int endAngle, int segments, Color color); // Draw circle sector outline void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2); // Draw a gradient-filled circle void DrawCircleV(Vector2 center, float radius, Color color); // Draw a color-filled circle (Vector version) void DrawCircleLines(int centerX, int centerY, float radius, Color color); // Draw circle outline void DrawEllipse(int centerX, int centerY, float radiusH, float radiusV, Color color); // Draw ellipse // Draw ellipse outline 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); // Draw ring void DrawRingLines(Vector2 center, float innerRadius, float outerRadius, int startAngle, int endAngle, int segments, Color color); // Draw ring outline void DrawRectangle(int posX, int posY, int width, int height, Color color); // Draw a color-filled rectangle void DrawRectangleV(Vector2 position, Vector2 size, Color color); // Draw a color-filled rectangle (Vector version) void DrawRectangleRec(Rectangle rec, Color color); // Draw a color-filled rectangle // Draw a color-filled rectangle with pro parameters void DrawRectanglePro(Rectangle rec, Vector2 origin, float rotation, Color color); void DrawRectangleGradientV(int posX, int posY, int width, int height, Color color1, Color color2); // Draw a vertical-gradient-filled rectangle void DrawRectangleGradientH(int posX, int posY, int width, int height, Color color1, Color color2); // Draw a horizontal-gradient-filled rectangle void DrawRectangleGradientEx(Rectangle rec, Color col1, Color col2, Color col3, Color col4); // Draw a gradient-filled rectangle with custom vertex colors void DrawRectangleLines(int posX, int posY, int width, int height, Color color); // Draw rectangle outline void DrawRectangleLinesEx(Rectangle rec, int lineThick, Color color); // Draw rectangle outline with extended parameters // Draw rectangle with rounded edges void DrawRectangleRounded(Rectangle rec, float roundness, int segments, Color color); void DrawRectangleRoundedLines(Rectangle rec, float roundness, int segments, int lineThick, Color color); // Draw rectangle with rounded edges outline void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color); // Draw a color-filled triangle (vertex in counter-clockwise order!) void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color); // Draw triangle outline (vertex in counter-clockwise order!) void DrawTriangleFan(Vector2 \*points, int numPoints, Color color); // Draw a triangle fan defined by points (first vertex is the center) void DrawTriangleStrip(Vector2 \*points, int pointsCount, Color color); // Draw a triangle strip defined by points void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color); // Draw a regular polygon (Vector version) void DrawPolyLines(Vector2 center, int sides, float radius, float rotation, Color color); // Draw a polygon outline of n sides // Basic shapes collision detection functions bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2); // Check collision between two rectangles bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2); // Check collision between two circles // Check collision between circle and rectangle bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec); Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2); // Get collision rectangle for two rectangles collision bool CheckCollisionPointRec(Vector2 point, Rectangle rec); // Check if point is inside rectangle bool CheckCollisionPointCircle(Vector2 point, Vector2 center, float radius); // Check if point is inside circle bool CheckCollisionPointTriangle(Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3); // Check if point is inside a triangle module: textures // Image loading functions // NOTE: This functions do not require GPU access Image LoadImage(const char \*fileName); // Load image from file into CPU memory (RAM) Image LoadImageEx(Color \*pixels, int width, int height); // Load image from Color array data (RGBA - 32bit) Image LoadImagePro(void \*data, int width, int height, int format); // Load image from raw data with parameters Image LoadImageRaw(const char \*fileName, int width, int height, int format, int headerSize); // Load image from RAW file data void UnloadImage(Image image); // Unload image from CPU memory (RAM) void ExportImage(Image image, const char \*fileName); // Export image data to file void ExportImageAsCode(Image image, const char \*fileName); // Export image as code file defining an array of bytes Color \*GetImageData(Image image); // Get pixel data from image as a Color struct array Vector4 \*GetImageDataNormalized(Image image); // Get pixel data from image as Vector4 array (float normalized) // Image generation functions Image GenImageColor(int width, int height, Color color); // Generate image: plain color Image GenImageGradientV(int width, int height, Color top, Color bottom); // Generate image: vertical gradient // Generate image: horizontal gradient Image GenImageGradientH(int width, int height, Color left, Color right); // Generate image: radial gradient 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); // Generate image: checked Image GenImageWhiteNoise(int width, int height, float factor); // Generate image: white noise Image GenImagePerlinNoise(int width, int height, int offsetX, int offsetY, float scale); // Generate image: perlin noise Image GenImageCellular(int width, int height, int tileSize); // Generate image: cellular algorithm. Bigger tileSize means bigger cells // Image manipulation functions Image ImageCopy(Image image); // Create an image duplicate (useful for transformations) Image ImageFromImage(Image image, Rectangle rec); // Create an image from another image piece // Create an image from text (default font) Image ImageText(const char \*text, int fontSize, Color color); // Create an image from text (custom sprite font) Image ImageTextEx(Font font, const char \*text, float fontSize, float spacing, Color tint); void ImageToPOT(Image \*image, Color fillColor); // Convert image to POT (power-of-two) void ImageFormat(Image \*image, int newFormat); // Convert image data to desired format void ImageAlphaMask(Image \*image, Image alphaMask); // Apply alpha mask to image void ImageAlphaClear(Image \*image, Color color, float threshold); // Clear alpha channel to desired color void ImageAlphaCrop(Image \*image, float threshold); // Crop image depending on alpha value void ImageAlphaPremultiply(Image \*image); // Premultiply alpha channel void ImageCrop(Image \*image, Rectangle crop); // Crop an image to a defined rectangle void ImageResize(Image \*image, int newWidth, int newHeight); // Resize image (Bicubic scaling algorithm) void ImageResizeNN(Image \*image, int newWidth,int newHeight); // Resize image (Nearest-Neighbor scaling algorithm) void ImageResizeCanvas(Image \*image, int newWidth, int newHeight, int offsetX, int offsetY, Color color); // Resize canvas and fill with color void ImageMipmaps(Image \*image); // Generate all mipmap levels for a provided image void ImageDither(Image \*image, int rBpp, int gBpp, int bBpp, int aBpp); // Dither image data to 16bpp or lower (Floyd–Steinberg dithering) void ImageFlipVertical(Image \*image); // Flip image vertically // Flip image horizontally void ImageFlipHorizontal(Image \*image); void ImageRotateCW(Image \*image); // Rotate image clockwise 90deg // Rotate image counter-clockwise 90deg void ImageRotateCCW(Image \*image); void ImageColorTint(Image \*image, Color color); // Modify image color: tint // Modify image color: invert void ImageColorInvert(Image \*image); void ImageColorGrayscale(Image \*image); // Modify image color: grayscale void ImageColorContrast(Image \*image, float contrast); // Modify image color: contrast (-100 to 100) void ImageColorBrightness(Image \*image, int brightness); // Modify image color: brightness (-255 to 255) void ImageColorReplace(Image \*image, Color color, Color replace); // Modify image color: replace color Color \*ImageExtractPalette(Image image, int maxPaletteSize, int \*extractCount); // Extract color palette from image to maximum size (memory should be freed) Rectangle GetImageAlphaBorder(Image image, float threshold); // Get image alpha border rectangle // Image drawing functions // NOTE: Image software-rendering functions (CPU) void ImageClearBackground(Image \*dst, Color color); // Clear image background with given color void ImageDrawPixel(Image \*dst, int posX, int posY, Color color); // Draw pixel within an image void ImageDrawPixelV(Image \*dst, Vector2 position, Color color); // Draw pixel within an image (Vector version) void ImageDrawLine(Image \*dst, int startPosX, int startPosY, int endPosX, int endPosY, Color color); // Draw line within an image void ImageDrawLineV(Image \*dst, Vector2 start, Vector2 end, Color color); // Draw line within an image (Vector version) void ImageDrawCircle(Image \*dst, int centerX, int centerY, int radius, Color color); // Draw circle within an image void ImageDrawCircleV(Image \*dst, Vector2 center, int radius, Color color); // Draw circle within an image (Vector version) void ImageDrawRectangle(Image \*dst, int posX, int posY, int width, int height, Color color); // Draw rectangle within an image // Draw rectangle within an image (Vector version) void ImageDrawRectangleV(Image \*dst, Vector2 position, Vector2 size, Color color); void ImageDrawRectangleRec(Image \*dst, Rectangle rec, Color color); // Draw rectangle within an image void ImageDrawRectangleLines(Image \*dst, Rectangle rec, int thick, Color color); // Draw rectangle lines within an image void ImageDraw(Image \*dst, Image src, Rectangle srcRec, Rectangle dstRec, Color tint); // Draw a source image within a destination image (tint applied to source) void ImageDrawText(Image \*dst, Vector2 position, const char \*text, int fontSize, Color color); // Draw text (default font) within an image (destination) void ImageDrawTextEx(Image \*dst, Vector2 position, Font font, const char \*text, float fontSize, float spacing, Color color); // Draw text (custom sprite font) within an image (destination) // Texture loading functions // NOTE: These functions require GPU access



// Encode text codepoint into utf8 text (memory must be freed!)

// Draw a line in 3D world space

// Draw cube (Vector version)

// Draw cube wires (Vector version)

// Draw cube

// Draw sphere

// Draw cube wires

// Draw a point in 3D space, actually a small line

// Get all codepoints in a string, codepoints count returned by parameters

// Encode codepoint into utf8 text (char array length returned as parameter)

// Returns next codepoint in a UTF8 encoded string; 0x3f('?') is returned on failure

// Get total number of characters (codepoints) in a UTF8 encoded string

char \*TextToUtf8(int \*codepoints, int length);

int \*GetCodepoints(const char \*text, int \*count);

// Basic geometric 3D shapes drawing functions

void DrawPoint3D(Vector3 position, Color color);

int GetNextCodepoint(const char \*text, int \*bytesProcessed);

const char \*CodepointToUtf8(int codepoint, int \*byteLength);

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

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

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

void DrawSphere(Vector3 centerPos, float radius, Color color);

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

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

void DrawCircle3D(Vector3 center, float radius, Vector3 rotationAxis, float rotationAngle, Color color); // Draw a circle in 3D world space

void DrawCubeTexture(Texture2D texture, Vector3 position, float width, float height, float length, Color color); // Draw cube textured

// UTF8 text strings management functions

int GetCodepointsCount(const char \*text);

module: models

struct Color;

struct Image;

struct Font;

struct Mesh;

struct Shader;

struct Model;

struct Ray;

struct Wave;

struct Sound;

struct Music;

struct Camera;

struct Texture;

void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color color); // Draw sphere with extended parameters void DrawSphereWires(Vector3 centerPos, float radius, int rings, int slices, Color color); // Draw sphere wires void DrawCylinder(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw α cylinder/cone wires void DrawPlane(Vector3 centerPos, Vector2 size, Color color); // Draw a plane XZ void DrawRay(Ray ray, Color color); // Draw a ray line void DrawGrid(int slices, float spacing); // Draw a grid (centered at (0, 0, 0)) void DrawGizmo(Vector3 position); // Draw simple gizmo // Model loading/unloading functions Model LoadModel(const char \*fileName); // Load model from files (meshes and materials) Model LoadModelFromMesh(Mesh mesh); // Load model from generated mesh (default material) void UnloadModel(Model model); // Unload model from memory (RAM and/or VRAM) // Mesh loading/unloading functions Mesh \*LoadMeshes(const char \*fileName, int \*meshCount); // Load meshes from model file void ExportMesh(Mesh mesh, const char \*fileName); // Export mesh data to file void UnloadMesh(Mesh mesh); // Unload mesh from memory (RAM and/or VRAM) // Material loading/unloading functions Material \*LoadMaterials(const char \*fileName, int \*materialCount); // Load materials from model file Material LoadMaterialDefault(void); // Load default material (Supports: DIFFUSE, SPECULAR, NORMAL maps) void UnloadMaterial(Material material); // Unload material from GPU memory (VRAM) void SetMaterialTexture(Material \*material, int mapType, Texture2D texture); // Set texture for a material map type (MAP\_DIFFUSE, MAP\_SPECULAR...) void SetModelMeshMaterial(Model \*model, int meshId, int materialId); // Set material for a mesh // Model animations loading/unloading functions ModelAnimation \*LoadModelAnimations(const char \*fileName, int \*animsCount); // Load model animations from file void UpdateModelAnimation(Model model, ModelAnimation anim, int frame); // Update model animation pose void UnloadModelAnimation(ModelAnimation anim); // Unload animation data // Check model animation skeleton match bool IsModelAnimationValid(Model model, ModelAnimation anim); // Mesh generation functions Mesh GenMeshPoly(int sides, float radius); // Generate polygonal mesh Mesh GenMeshPlane(float width, float length, int resX, int resZ); // Generate plane mesh (with subdivisions) Mesh GenMeshCube(float width, float height, float length); // Generate cuboid mesh Mesh GenMeshSphere(float radius, int rings, int slices); // Generate sphere mesh (standard sphere) Mesh GenMeshHemiSphere(float radius, int rings, int slices); // Generate half-sphere mesh (no bottom cap) Mesh GenMeshCylinder(float radius, float height, int slices); // Generate cylinder mesh Mesh GenMeshTorus(float radius, float size, int radSeg, int sides); // Generate torus mesh Mesh GenMeshKnot(float radius, float size, int radSeg, int sides); // Generate trefoil knot mesh Mesh GenMeshHeightmap(Image heightmap, Vector3 size); // Generate heightmap mesh from image data Mesh GenMeshCubicmap(Image cubicmap, Vector3 cubeSize); // Generate cubes-based map mesh from image data // Mesh manipulation functions BoundingBox MeshBoundingBox(Mesh mesh); // Compute mesh bounding box limits void MeshTangents(Mesh \*mesh); // Compute mesh tangents void MeshBinormals(Mesh \*mesh); // Compute mesh binormals // Model drawing functions void DrawModel(Model model, Vector3 position, float scale, Color tint); // Draw a model (with texture if set) void DrawModelEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model with extended parameters void DrawModelWires(Model model, Vector3 position, float scale, Color tint); // Draw a model wires (with texture if set) void DrawModelWiresEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model wires (with texture if set) with extended parameters void DrawBoundingBox(BoundingBox box, Color color); // Draw bounding box (wires) void DrawBillboard(Camera camera, Texture2D texture, Vector3 center, float size, Color tint); // Draw a billboard texture void DrawBillboardRec(Camera camera, Texture2D texture, Rectangle sourceRec, Vector3 center, float size, Color tint); // Draw a billboard texture defined by sourceRec // Collision detection functions bool CheckCollisionSpheres(Vector3 centerA, float radiusA, Vector3 centerB, float radiusB); // Detect collision between two spheres bool CheckCollisionBoxes(BoundingBox box1, BoundingBox box2); // Detect collision between two bounding boxes bool CheckCollisionBoxSphere(BoundingBox box, Vector3 center, float radius); // Detect collision between box and sphere bool CheckCollisionRaySphere(Ray ray, Vector3 center, float radius); // Detect collision between ray and sphere bool CheckCollisionRaySphereEx(Ray ray, Vector3 center, float radius, Vector3 \*collisionPoint); // Detect collision between ray and sphere, returns collision point bool CheckCollisionRayBox(Ray ray, BoundingBox box); // Detect collision between ray and box RayHitInfo GetCollisionRayModel(Ray ray, Model model); // Get collision info between ray and model RayHitInfo GetCollisionRayTriangle(Ray ray, Vector3 p1, Vector3 p2, Vector3 p3); // Get collision info between ray and triangle RayHitInfo GetCollisionRayGround(Ray ray, float groundHeight); // Get collision info between ray and ground plane (Y-normal plane) module: shaders (rlgl) // Shader loading/unloading functions char \*LoadText(const char \*fileName); // Load chars array from text file Shader LoadShader(const char \*vsFileName, const char \*fsFileName); // Load shader from files and bind default locations Shader LoadShaderCode(char \*vsCode, char \*fsCode); // Load shader from code strings and bind default locations void UnloadShader(Shader shader); // Unload shader from GPU memory (VRAM) Shader GetShaderDefault(void); // Get default shader Texture2D GetTextureDefault(void); // Get default texture // Get texture to draw shapes Texture2D GetShapesTexture(void); Rectangle GetShapesTextureRec(void); // Get texture rectangle to draw shapes void SetShapesTexture(Texture2D texture, Rectangle source); // Define default texture used to draw shapes // Shader configuration functions int GetShaderLocation(Shader shader, const char \*uniformName); // Get shader uniform location void SetShaderValue(Shader shader, int uniformLoc, const void \*value, int uniformType); // Set shader uniform value void SetShaderValueV(Shader shader, int uniformLoc, const void \*value, int uniformType, int count); // Set shader uniform value vector void SetShaderValueMatrix(Shader shader, int uniformLoc, Matrix mat); // Set shader uniform value (matrix 4x4) void SetShaderValueTexture(Shader shader, int uniformLoc, Texture2D texture); // Set shader uniform value for texture void SetMatrixProjection(Matrix proj); // Set a custom projection matrix (replaces internal projection matrix) void SetMatrixModelview(Matrix view); // Set a custom modelview matrix (replaces internal modelview matrix) Matrix GetMatrixModelview(); // Get internal modelview matrix Matrix GetMatrixProjection(void); // Get internal projection matrix // Shading begin/end functions void BeginShaderMode(Shader shader); // Begin custom shader drawing void EndShaderMode(void); // End custom shader drawing (use default shader) void BeginBlendMode(int mode); // Begin blending mode (alpha, additive, multiplied) void EndBlendMode(void); // End blending mode (reset to default: alpha blending) // VR control functions void InitVrSimulator(void); // Init VR simulator for selected device parameters void CloseVrSimulator(void); // Close VR simulator for current device void UpdateVrTracking(Camera \*camera); // Update VR tracking (position and orientation) and camera void SetVrConfiguration(VrDeviceInfo info, Shader distortion); // Set stereo rendering configuration parameters bool IsVrSimulatorReady(void); // Detect if VR simulator is ready void ToggleVrMode(void); // Enable/Disable VR experience



