

module: core

```

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

// 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 is currently fullscreen
// Check if window is currently hidden (only PLATFORM_DESKTOP)
// Check if window is currently minimized (only PLATFORM_DESKTOP)
// Check if window is currently maximized (only PLATFORM_DESKTOP)
// Check if window is currently focused (only PLATFORM_DESKTOP)
// Check if window has been resized last frame
// Check if one specific window flag is enabled
// Set window configuration state using flags
// Clear window configuration state flags
// Toggle window state: fullscreen/windowed (only PLATFORM_DESKTOP)
// Set window state: maximized, if resizable (only PLATFORM_DESKTOP)
// Set window state: minimized, if resizable (only PLATFORM_DESKTOP)
// Set window state: not minimized/maximized (only PLATFORM_DESKTOP)
// Set icon for window (only PLATFORM_DESKTOP)
// Set title for window (only PLATFORM_DESKTOP)
// Set window position on screen (only PLATFORM_DESKTOP)
// Set monitor for the current window (fullscreen mode)
// Set window minimum dimensions (for FLAG_WINDOW_RESIZABLE)
// Set window dimensions
// Get native window handle
// Get current screen width
// Get current screen height
// Get number of connected monitors
// Get current connected monitor
// Get specified monitor position
// Get specified monitor width (max available by monitor)
// Get specified monitor height (max available by monitor)
// Get specified monitor physical width in millimetres
// Get specified monitor physical height in millimetres
// Get specified monitor refresh rate
// Get window position XY on monitor
// Get window scale DPI factor
// Get the human-readable, UTF-8 encoded name of the primary monitor
// Set clipboard text content
// Get clipboard text content

// 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 screen

// Drawing-related functions
void ClearBackground(Color color);
void BeginDrawing(void);
void EndDrawing(void);
void BeginMode2D(Camera2D camera);
void EndMode2D(void);
void BeginMode3D(Camera3D camera);
void EndMode3D(void);
void BeginTextureMode(RenderTexture2D target);
void EndTextureMode(void);
void BeginShaderMode(Shader shader);
void EndShaderMode(void);
void BeginBlendMode(int mode);
void EndBlendMode(void);
void BeginScissorMode(int x, int y, int width, int height);
void EndScissorMode(void);
void BeginVrStereoMode(VrStereoConfig config);
void EndVrStereoMode(void);

// Set background color (framebuffer clear color)
// Setup canvas (framebuffer) to start drawing
// End canvas drawing and swap buffers (double buffering)
// Begin 2D mode with custom camera (2D)
// Ends 2D mode with custom camera
// Begin 3D mode with custom camera (3D)
// Ends 3D mode and returns to default 2D orthographic mode
// Begin drawing to render texture
// Ends drawing to render texture
// Begin custom shader drawing
// End custom shader drawing (use default shader)
// Begin blending mode (alpha, additive, multiplied, subtract, custom)
// End blending mode (reset to default: alpha blending)
// Begin scissor mode (define screen area for following drawing)
// End scissor mode
// Begin stereo rendering (requires VR simulator)
// End stereo rendering (requires VR simulator)

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

// Load VR stereo config for VR simulator device parameters
// Unload VR stereo config

// Shader management functions
// NOTE: Shader functionality is not available on OpenGL 1.1
Shader LoadShader(const char *vsFileName, const char *fsFileName);
Shader LoadShaderFromMemory(const char *vsCode, const char *fsCode);
int GetShaderLocation(Shader shader, const char *uniformName);
int GetShaderLocationAttrib(Shader shader, const char *attribName);
void SetShaderValue(Shader shader, int locIndex, const void *value, int uniformType);
void SetShaderValueV(Shader shader, int locIndex, const void *value, int uniformType, int count);
void SetShaderValueMatrix(Shader shader, int locIndex, Matrix mat);
void SetShaderValueTexture(Shader shader, int locIndex, Texture2D texture);
void UnloadShader(Shader shader);

// Load shader from files and bind default locations
// Load shader from code strings and bind default locations
// 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 (sampler2d)
// Unload shader from GPU memory (VRAM)

// Screen-space-related functions
Ray GetMouseRay(Vector2 mousePosition, Camera camera);
Matrix GetCameraMatrix(Camera camera);
Matrix GetCameraMatrix2D(Camera2D camera);
Vector2 GetWorldToScreen(Vector3 position, Camera camera);
Vector2 GetWorldToScreenEx(Vector3 position, Camera camera, int width, int height);
Vector2 GetWorldToScreen2D(Vector2 position, Camera2D camera);
Vector2 GetScreenToWorld2D(Vector2 position, Camera2D camera);

// Get a ray trace from mouse position
// Get camera transform matrix (view matrix)
// Get camera 2d transform matrix
// Get the screen space position for a 3d world space position
// Get size position for a 3d world space position
// Get the screen space position for a 2d camera world space position
// Get 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)
// Get current FPS
// Get time in seconds for last frame drawn (delta time)
// Get elapsed time in seconds since InitWindow()

// Misc. functions
int GetRandomValue(int min, int max);
void SetRandomSeed(unsigned int seed);
void TakeScreenshot(const char *fileName);
void SetConfigFlags(unsigned int flags);

// Get a random value between min and max (both included)
// Set the seed for the random number generator
// Takes a screenshot of current screen (filename extension defines format)
// Setup init configuration flags (view FLAGS)

void TraceLog(int logLevel, const char *text, ...);
void SetTraceLogLevel(int logLevel);
void *MemAlloc(int size);
void *MemRealloc(void *ptr, int size);
void MemFree(void *ptr);

// Show trace log messages (LOG_DEBUG, LOG_INFO, LOG_WARNING, LOG_ERROR...)
// Set the current threshold (minimum) log level
// Internal memory allocator
// Internal memory reallocator
// Internal memory free

// Set custom callbacks
// WARNING: Callbacks setup is intended for advance users
void SetTraceLogCallback(TraceLogCallback callback);
void SetLoadFileDataCallback(LoadFileDataCallback callback);
void SetSaveFileDataCallback(SaveFileDataCallback callback);
void SetLoadFileTextCallback(LoadFileTextCallback callback);
void SetSaveFileTextCallback(SaveFileTextCallback callback);

// Set custom trace log
// Set custom file binary data loader
// Set custom file binary data saver
// Set custom file text data loader
// Set custom file text data saver

// Files management functions
unsigned char *LoadFileData(const char *fileName, unsigned int *bytesRead);
void UnloadFileData(unsigned char *data);
bool SaveFileData(const char *fileName, void *data, unsigned int bytesToWrite);
char *LoadFileText(const char *fileName);
void UnloadFileText(char *text);
bool SaveFileText(const char *fileName, char *text);
bool FileExists(const char *fileName);
bool DirectoryExists(const char *dirPath);
bool IsFileExtension(const char *fileName, const char *ext);
const char *GetFileExtension(const char *fileName);
const char *GetFileName(const char *filePath);
const char *GetFileNameWithoutExt(const char *filePath);
const char *GetDirectoryPath(const char *filePath);
const char *GetPrevDirectoryPath(const char *dirPath);
const char *GetWorkingDirectory(void);
char **GetDirectoryFiles(const char *dirPath, int *count);
void ClearDirectoryFiles(void);
bool ChangeDirectory(const char *dir);
bool IsFileDropped(void);
char **GetDroppedFiles(int *count);
void ClearDroppedFiles(void);
long GetFileModTime(const char *fileName);

// Load file data as byte array (read)
// Unload file data allocated by LoadFileData()
// Save data to file from byte array (write), returns true on success
// Load text data from file (read), returns a '\0' terminated string
// Unload file text data allocated by LoadFileText()
// Save text data to file (write), string must be '\0' terminated, returns true on success
// Check if file exists
// Check if a directory path exists
// Check file extension (including point: .png, .wav)
// Get pointer to extension for a filename string (includes dot: '.png')
// Get pointer to filename for a path string
// Get filename string without extension (uses static string)
// Get full path for a given fileName with path (uses static string)
// Get previous directory path for a given path (uses static string)
// Get current working directory (uses static string)
// Get filenames in a directory path (memory should be freed)
// Clear directory files paths buffers (free memory)
// Change working directory, return true on success
// Check if a file has been dropped into window
// Get dropped files names (memory should be freed)
// Clear dropped files paths buffer (free memory)
// Get file modification time (last write time)

```

```

// Compression/Encoding functionality
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)
char *EncodeDataBase64(const unsigned char *data, int dataLength, int *outputLength); // Encode data to Base64 string
unsigned char *DecodeDataBase64(unsigned char *data, int *outputLength); // Decode Base64 string data

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

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

// Input-related functions: keyboard
bool IsKeyPressed(int key); // Check if a key has been pressed once
bool IsKeyDown(int key); // Check if a key is being pressed
bool IsKeyReleased(int key); // Check if a key has been released once
bool IsKeyUp(int key); // Check if a key is NOT being pressed
void SetExitKey(int key); // Set a custom key to exit program (default is ESC)
int GetKeyPressed(void); // Get key pressed (keycode), call it multiple times for keys queued, returns 0 when the queue is empty
int GetCharPressed(void); // Get char pressed (unicode), call it multiple times for chars queued, returns 0 when the queue is empty

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

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

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

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

// Camera System Functions (Module: rcamera)
void SetCameraMode(Camera camera, int mode); // Set camera mode (multiple camera modes available)
void UpdateCamera(Camera *camera); // Update camera position for selected mode

void SetCameraPanControl(int keyPan); // Set camera pan key to combine with mouse movement (free camera)
void SetCameraAltControl(int keyAlt); // Set camera alt key to combine with mouse movement (free camera)
void SetCameraSmoothZoomControl(int keySmoothZoom); // Set camera smooth zoom key to combine with mouse (free camera)
void SetCameraMoveControls(int keyFront, int keyBack, int keyRight, int keyLeft, int keyUp, int keyDown); // Set camera move controls (1st person and 3rd person cameras)

```

module: shapes

```

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

// Basic shapes drawing functions
void DrawPixel(int posX, int posY, Color color); // Draw a pixel
void DrawPixelV(Vector2 position, Color color); // Draw a pixel (Vector version)
void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color); // Draw a line
void DrawLineV(Vector2 startPos, Vector2 endPos, Color color); // Draw a line (Vector version)
void DrawLineEx(Vector2 startPos, Vector2 endPos, float thick, Color color); // Draw a line defining thickness
void DrawLineBezier(Vector2 startPos, Vector2 endPos, float thick, Color color); // Draw a line using cubic-bezier curves in-out
void DrawLineBezierQuad(Vector2 startPos, Vector2 endPos, Vector2 controlPos, float thick, Color color); // Draw line using quadratic bezier curves with a control point
void DrawLineBezierCubic(Vector2 startPos, Vector2 endPos, Vector2 startControlPos, Vector2 endControlPos, float thick, Color color); // Draw line using cubic bezier curves with two control points
void DrawLineStrip(Vector2 *points, int pointsCount, Color color); // Draw lines sequence
void DrawCircle(int centerX, int centerY, float radius, Color color); // Draw a color-filled circle
void DrawCircleSector(Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color); // Draw a piece of a circle
void DrawCircleSectorLines(Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color); // Draw circle sector outline
void DrawCircleGradient(int centerX, int centerY, float radius, Color color1, Color color2); // Draw a gradient-filled circle
void DrawCircleV(Vector2 center, float radius, Color color); // Draw a color-filled circle (Vector version)
void DrawCircleLines(int centerX, int centerY, float radius, Color color); // Draw circle outline
void DrawEllipse(int centerX, int centerY, float radiusH, float radiusV, Color color); // Draw ellipse
void DrawEllipseLines(int centerX, int centerY, float radiusH, float radiusV, Color color); // Draw ellipse outline
void DrawRing(Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color); // Draw ring
void DrawRingLines(Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color); // Draw ring outline
void DrawRectangle(int posX, int posY, int width, int height, Color color); // Draw a color-filled rectangle
void DrawRectangleV(Vector2 position, Vector2 size, Color color); // Draw a color-filled rectangle (Vector version)
void DrawRectangleRec(Rectangle rec, Color color); // Draw a color-filled rectangle
void DrawRectanglePro(Rectangle rec, Vector2 origin, float rotation, Color color); // Draw a color-filled rectangle with pro parameters
void DrawRectangleGradientV(int posX, int posY, int width, int height, Color color1, Color color2); // Draw a vertical-gradient-filled rectangle
void DrawRectangleGradientH(int posX, int posY, int width, int height, Color color1, Color color2); // Draw a horizontal-gradient-filled rectangle
void DrawRectangleGradientEx(Rectangle rec, Color col1, Color col2, Color col3, Color col4); // Draw a gradient-filled rectangle with custom vertex colors
void DrawRectangleLines(int posX, int posY, int width, int height, Color color); // Draw rectangle outline
void DrawRectangleLinesEx(Rectangle rec, int lineThick, Color color); // Draw rectangle outline with extended parameters
void DrawRectangleRounded(Rectangle rec, float roundness, int segments, Color color); // Draw rectangle with rounded edges
void DrawRectangleRoundedLines(Rectangle rec, float roundness, int segments, int lineThick, Color color); // Draw rectangle with rounded edges outline
void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color); // Draw a color-filled triangle (vertex in counter-clockwise order!)
void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color); // Draw triangle outline (vertex in counter-clockwise order!)
void DrawTriangleFan(Vector2 *points, int pointsCount, 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
void DrawPolyLinesEx(Vector2 center, int sides, float radius, float rotation, float lineThick, Color color); // Draw a polygon outline of n sides with extended parameters

// Basic shapes collision detection functions
bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2); // Check collision between two rectangles
bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2); // Check collision between two circles
bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec); // Check collision between circle and rectangle
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
bool CheckCollisionLines(Vector2 startPos1, Vector2 endPos1, Vector2 startPos2, Vector2 endPos2, Vector2 *collisionPoint); // Check the collision between two lines defined by endpoints
bool CheckCollisionPointLine(Vector2 point, Vector2 p1, Vector2 p2, int threshold); // Check if point belongs to line created between two points [p1] and [p2]
Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2); // Get collision rectangle for two rectangles collision

```

module: textures

```

// Image loading functions
// NOTE: This functions do not require GPU access
Image LoadImage(const char *fileName); // Load image from file into CPU memory (RAM)
Image LoadImageRaw(const char *fileName, int width, int height, int format, int headerSize); // Load image from RAW file data
Image LoadImageAnim(const char *fileName, int *frames); // Load image sequence from file (frames appended to image.data)
Image LoadImageFromMemory(const char *fileName, const unsigned char *fileData, int dataSize); // Load image from memory buffer
Image LoadImageFromTexture(Texture2D texture); // Load image from GPU texture data
Image LoadImageFromScreen(void); // Load image from screen buffer and (screenshot)
void UnloadImage(Image image); // Unload image from CPU memory (RAM)
bool ExportImage(Image image, const char *fileName); // Export image data to file, returns true on success
bool ExportImageAsCode(Image image, const char *fileName); // Export image as code file defining an array of bytes, returns true on success

```

```

// Image generation functions
Image GenImageColor(int width, int height, Color color); // Generate image: plain color
Image GenImageGradientV(int width, int height, Color top, Color bottom); // Generate image: vertical gradient
Image GenImageGradientH(int width, int height, Color left, Color right); // Generate image: horizontal gradient
Image GenImageGradientRadial(int width, int height, float density, Color inner, Color outer); // Generate image: radial gradient
Image GenImageChecked(int width, int height, int checksX, int checksY, Color col1, Color col2); // Generate image: checked
Image GenImageWhiteNoise(int width, int height, float factor); // Generate image: white noise
Image 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
Image ImageText(const char *text, int fontSize, Color color); // Create an image from text (default font)
Image ImageTextEx(Font font, const char *text, float fontSize, float spacing, Color tint); // Create an image from text (custom sprite font)
void ImageFormat(Image *image, int newFormat); // Convert image data to desired format
void ImageToPOT(Image *image, Color fill); // Convert image to POT (power-of-two)
void ImageCrop(Image *image, Rectangle crop); // Crop an image to a defined rectangle
void ImageAlphaCrop(Image *image, float threshold); // Crop image depending on alpha value
void ImageAlphaClear(Image *image, Color color, float threshold); // Clear alpha channel to desired color
void ImageAlphaMask(Image *image, Image alphaMask); // Apply alpha mask to image
void ImageAlphaPremultiply(Image *image); // Premultiply alpha channel
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 fill); // 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
void ImageFlipHorizontal(Image *image); // Flip image horizontally
void ImageRotateCW(Image *image); // Rotate image clockwise 90deg
void ImageRotateCCW(Image *image); // Rotate image counter-clockwise 90deg
void ImageColorTint(Image *image, Color color); // Modify image color: tint
void ImageColorInvert(Image *image); // Modify image color: invert
void ImageColorGrayscale(Image *image); // Modify image color: grayscale
void ImageColorContrast(Image *image, float contrast); // Modify image color: contrast (-100 to 100)
void ImageColorBrightness(Image *image, int brightness); // Modify image color: brightness (-255 to 255)
void ImageColorReplace(Image *image, Color color, Color replace); // Modify image color: replace color
Color *LoadImageColors(Image image); // Load color data from image as a Color array (RGBA - 32bit)
Color *LoadImagePalette(Image image, int maxPaletteSize, int *colorsCount); // Load colors palette from image as a Color array (RGBA - 32bit)
void UnloadImageColors(Color *colors); // Unload color data loaded with LoadImageColors()
void UnloadImagePalette(Color *colors); // Unload colors palette loaded with LoadImagePalette()
Rectangle GetImageAlphaBorder(Image image, float threshold); // Get image alpha border rectangle
Color GetImageColor(Image image, int x, int y); // Get image pixel color at (x, y) position

// 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
void ImageDrawRectangleV(Image *dst, Vector2 position, Vector2 size, Color color); // Draw rectangle within an image (Vector version)
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, const char *text, int posX, int posY, int fontSize, Color color); // Draw text (using default 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 (destination)

// Texture loading functions
// NOTE: These functions require GPU access
Texture2D LoadTexture(const char *fileName); // Load texture from file into GPU memory (VRAM)
Texture2D LoadTextureFromImage(Image image); // Load texture from image data
TextureCubemap LoadTextureCubemap(Image image, int layout); // Load cubemap from image, multiple image cubemap layouts supported
RenderTexture2D LoadRenderTexture(int width, int height); // Load texture for rendering (framebuffer)
void UnloadTexture(Texture2D texture); // Unload texture from GPU memory (VRAM)
void UnloadRenderTexture(RenderTexture2D target); // Unload render texture from GPU memory (VRAM)
void UpdateTexture(Texture2D texture, const void *pixels); // Update GPU texture with new data
void UpdateTextureRec(Texture2D texture, Rectangle rec, const void *pixels); // Update GPU texture rectangle with new data

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

// Texture drawing functions
void DrawTexture(Texture2D texture, int posX, int posY, Color tint); // Draw a Texture2D
void DrawTextureV(Texture2D texture, Vector2 position, Color tint); // Draw a Texture2D with position defined as Vector2
void DrawTextureEx(Texture2D texture, Vector2 position, float rotation, float scale, Color tint); // Draw a Texture2D with extended parameters
void DrawTextureRec(Texture2D texture, Rectangle source, Vector2 position, Color tint); // Draw a part of a texture defined by a rectangle
void DrawTextureQuad(Texture2D texture, Vector2 tiling, Vector2 offset, Rectangle quad, Color tint); // Draw texture quad with tiling and offset parameters
void DrawTextureTiled(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, float scale, Color tint); // Draw part of a texture (defined by a rectangle) tiled onto another rectangle
void DrawTexturePro(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, Color tint); // Draw a part of a texture defined by a rectangle onto another rectangle
void DrawTextureNPatch(Texture2D texture, NPatchInfo nPatchInfo, Rectangle dest, Vector2 origin, float rotation, Color tint); // Draws a texture (or part of it) that stretches and rotates correctly
void DrawTexturePoly(Texture2D texture, Vector2 center, Vector2 *points, Vector2 *texcoords, int pointsCount, Color tint); // Draw a textured polygon

// Color/pixel related functions
Color Fade(Color color, float alpha); // Returns color with alpha applied, alpha goes from 0.0f to 1.0f
int ColorToInt(Color color); // Returns hexadecimal value for a Color
Vector4 ColorNormalize(Color color); // Returns Color normalized as float [0..1]
Color ColorFromNormalized(Vector4 normalized); // Returns Color from normalized values [0..1]
Vector3 ColorToHSV(Color color); // Returns HSV values for a Color, hue [0..360], saturation/value [0..1]
Color ColorFromHSV(float hue, float saturation, float value); // Returns a Color from HSV values, hue [0..360], saturation/value [0..1]
Color ColorAlpha(Color color, float alpha); // Returns color with alpha applied, alpha goes from 0.0f to 1.0f
Color ColorAlphaBlend(Color dst, Color src, Color tint); // Returns src alpha-blended into dst color with tint
Color GetColor(unsigned int hexValue); // Get Color structure from hexadecimal value
Color GetPixelColor(void *srcPtr, int format); // Get Color from a source pixel pointer of certain format
void SetPixelColor(void *dstPtr, Color color, int format); // Set color formatted into destination pixel pointer
int GetPixelDataSize(int width, int height, int format); // Get pixel data size in bytes for certain format

```

module: text

```

// Font loading/unloading functions
Font GetFontDefault(void); // Get the default Font
Font LoadFont(const char *fileName); // Load font from file into GPU memory (VRAM)
Font LoadFontEx(const char *fileName, int fontSize, int *fontChars, int glyphCount); // Load font from file with extended parameters
Font LoadFontFromImage(Image image, Color key, int firstChar); // Load font from image
Font LoadFontFromMemory(const char *filePath, const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int glyphCount); // Load font from memory buffer, file path is optional
GlyphInfo *LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *fontChars, int glyphCount, int type); // Load font data for further use
Image GenImageFontAtlas(const GlyphInfo *chars, Rectangle **recs, int glyphCount, int fontSize, int padding, int packMethod); // Generate image font atlas using chars info
void UnloadFontData(GlyphInfo *chars, int glyphCount); // Unload font chars info data (RAM)
void UnloadFont(Font font); // Unload Font from GPU memory (VRAM)

// Text drawing functions
void DrawFPS(int posX, int posY); // Draw current FPS
void DrawText(const char *text, int posX, int posY, int fontSize, Color color); // Draw text (using default font)
void DrawTextEx(Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint); // Draw text using font and additional parameters
void DrawTextPro(Font font, const char *text, Vector2 position, float rotation, float fontSize, float spacing, Color tint); // Draw text using Font and pro parameters
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); // Measure string width for default font
Vector2 MeasureTextEx(Font font, const char *text, float fontSize, float spacing); // Measure string size for Font
int GetGlyphIndex(Font font, int codepoint); // Get glyph index position in font for a codepoint (unicode character), fallback to '?' if not found
GlyphInfo GetGlyphInfo(Font font, int codepoint); // Get glyph font info data for a codepoint (unicode character), fallback to '?' if not found
Rectangle GetGlyphAtlasRec(Font font, int codepoint); // Get glyph rectangle in font atlas for a codepoint (unicode character), fallback to '?' if not found

// Text codepoints management functions (unicode characters)
int *LoadCodepoints(const char *text, int *count); // Load all codepoints from a UTF-8 text string, codepoints count returned by parameter
void UnloadCodepoints(int *codepoints); // Unload codepoints data from memory
int GetCodepointCount(const char *text); // Get total number of codepoints in a UTF-8 encoded string
int GetCodepoint(const char *text, int *bytesProcessed); // Get next codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
const char *CodepointToUTF8(int codepoint, int *byteSize); // Encode one codepoint into UTF-8 byte array (array length returned as parameter)
char *TextCodepointsToUTF8(int *codepoints, int length); // Encode text as codepoints array into UTF-8 text string (WARNING: memory must be freed!)

// Text strings management functions (no utf8 strings, only byte chars)
// NOTE: Some strings allocate memory internally for returned strings, just be careful!
int TextCopy(char *dst, const char *src); // Copy one string to another, returns bytes copied
bool TextIsEqual(const char *text1, const char *text2); // Check if two text string are equal
unsigned int TextLength(const char *text); // Get text length, checks for '\0' ending
const char *TextFormat(const char *text, ...); // Text formatting with variables (sprintf style)
const char *TextSubtext(const char *text, int position, int length); // Get a piece of a text string
char *TextReplace(char *text, const char *replace, const char *by); // Replace text string (memory must be freed!)
char *TextInsert(const char *text, const char *insert, int position); // Insert text in a position (memory must be freed!)
const char *TextJoin(const char **textList, int count, const char *delimiter); // Join text strings with delimiter
const char **TextSplit(const char *text, char delimiter, int *count); // Split text into multiple strings
void TextAppend(char *text, const char *append, int *position); // Append text at specific position and move cursor!
int TextFindIndex(const char *text, const char *find); // Find first text occurrence within a string
const char *TextToUpper(const char *text); // Get upper case version of provided string
const char *TextToLower(const char *text); // Get lower case version of provided string

```

```
const char *TextToPascal(const char *text);
int TextToInteger(const char *text);
```

```
// Get Pascal case notation version of provided string
// Get integer value from text (negative values not supported)
```

module: models

```
// Basic geometric 3D shapes drawing functions
void DrawLine3D(Vector3 startPos, Vector3 endPos, Color color); // Draw a line in 3D world space
void DrawPoint3D(Vector3 position, Color color); // Draw a point in 3D space, actually a small line
void DrawCircle3D(Vector3 center, float radius, Vector3 rotationAxis, float rotationAngle, Color color); // Draw a circle in 3D world space
void DrawTriangle3D(Vector3 v1, Vector3 v2, Vector3 v3, Color color); // Draw a color-filled triangle (vertex in counter-clockwise order!)
void DrawTriangleStrip3D(Vector3 *points, int pointsCount, Color color); // Draw a triangle strip defined by points
void DrawCube(Vector3 position, float width, float height, float length, Color color); // Draw cube
void DrawCubeV(Vector3 position, Vector3 size, Color color); // Draw cube (Vector version)
void DrawCubeWires(Vector3 position, float width, float height, float length, Color color); // Draw cube wires
void DrawCubeWiresV(Vector3 position, Vector3 size, Color color); // Draw cube wires (Vector version)
void DrawCubeTexture(Texture2D texture, Vector3 position, float width, float height, float length, Color color); // Draw cube textured
void DrawCubeTextureRec(Texture2D texture, Rectangle source, Vector3 position, float width, float height, float length, Color color); // Draw cube with a region of a texture
void DrawSphere(Vector3 centerPos, float radius, Color color); // Draw sphere
void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color color); // Draw sphere with extended parameters
void DrawSphereWires(Vector3 centerPos, float radius, int rings, int slices, Color color); // Draw sphere wires
void DrawCylinder(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone
void DrawCylinderEx(Vector3 startPos, Vector3 endPos, float startRadius, float endRadius, int sides, Color color); // Draw a cylinder with base at startPos and top at endPos
void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone wires
void DrawCylinderWiresEx(Vector3 startPos, Vector3 endPos, float startRadius, float endRadius, int sides, Color color); // Draw a cylinder wires with base at startPos and top at endPos
void DrawPlane(Vector3 centerPos, Vector2 size, Color color); // Draw a plane XZ
void DrawRay(Ray ray, Color color); // Draw a ray line
void DrawGrid(int slices, float spacing); // Draw a grid (centered at (0, 0, 0))

// Model 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 (including meshes) from memory (RAM and/or VRAM)
void UnloadModelKeepMeshes(Model model); // Unload model (but not meshes) from memory (RAM and/or VRAM)
BoundingBox GetModelBoundingBox(Model model); // Compute model bounding box limits (considers all meshes)

// Model drawing functions
void DrawModel(Model model, Vector3 position, float scale, Color tint); // Draw a model (with texture if set)
void DrawModelEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model with 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 position, float size, Color tint); // Draw a billboard texture
void DrawBillboardRec(Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector2 size, Color tint); // Draw a billboard texture defined by source
void DrawBillboardPro(Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector3 up, Vector2 size, Vector2 origin, float rotation, Color tint); // Draw a billboard texture defined by source and rotation

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

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

// Material loading/unloading functions
Material *LoadMaterials(const char *fileName, int *materialCount); // Load materials from model file
Material LoadMaterialDefault(void); // Load default material (Supports: DIFFUSE, SPECULAR, 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 (MATERIAL_MAP_DIFFUSE, MATERIAL_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, unsigned int *animCount); // Load model animations from file
void UpdateModelAnimation(Model model, ModelAnimation anim, int frame); // Update model animation pose
void UnloadModelAnimation(ModelAnimation anim); // Unload animation data
void UnloadModelAnimations(ModelAnimation* animations, unsigned int count); // Unload animation array data
bool IsModelAnimationValid(Model model, ModelAnimation anim); // Check model animation skeleton match

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

module: audio

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

// Wave/Sound loading/unloading functions
Wave LoadWave(const char *fileName); // Load wave data from file
Wave LoadWaveFromMemory(const char *fileName, const unsigned char *fileData, int dataSize); // Load wave from memory buffer
Sound LoadSound(const char *fileName); // Load sound from file
Sound LoadSoundFromWave(Wave wave); // Load sound from wave data
void UpdateSound(Sound sound, const void *data, int samplesCount); // Update sound buffer with new data
void UnloadWave(Wave wave); // Unload wave data
void UnloadSound(Sound sound); // Unload sound
bool ExportWave(Wave wave, const char *fileName); // Export wave data to file, returns true on success
bool ExportWaveAsCode(Wave wave, const char *fileName); // Export wave sample data to code (.h), returns true on success

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

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

// AudioStream management functions
```

```

AudioStream InitAudioStream(unsigned int sampleRate, unsigned int sampleSize, unsigned int channels); // Init audio stream (to stream raw audio pcm data)
void UpdateAudioStream(AudioStream stream, const void *data, int samplesCount); // Update audio stream buffers with data
void CloseAudioStream(AudioStream stream); // Close audio stream and free memory
bool IsAudioStreamProcessed(AudioStream stream); // Check if any audio stream buffers requires refill
void PlayAudioStream(AudioStream stream); // Play audio stream
void PauseAudioStream(AudioStream stream); // Pause audio stream
void ResumeAudioStream(AudioStream stream); // Resume audio stream
bool IsAudioStreamPlaying(AudioStream stream); // Check if audio stream is playing
void StopAudioStream(AudioStream stream); // Stop audio stream
void SetAudioStreamVolume(AudioStream stream, float volume); // Set volume for audio stream (1.0 is max level)
void SetAudioStreamPitch(AudioStream stream, float pitch); // Set pitch for audio stream (1.0 is base level)
void SetAudioStreamBufferSizeDefault(int size); // Default size for new audio streams

```

structs

```

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

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

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

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

struct VrDeviceInfo; // VR device parameters
struct VrStereoConfig; // VR Stereo rendering configuration for simulator

```

colors

```

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

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

```