My Project

AUTHOR
Versão 1.0

Sumário

Table of contents

Índice das Estruturas de Dados

Estruturas de Dados

Aqui estão as estruturas de dados, uniões e suas respectivas descrições:

Índice dos Arquivos

Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Estruturas

Referência da Estrutura AudioStream

```
#include <raylib.h>
```

Campos de Dados

- **rAudioBuffer** * **buffer**
- **rAudioProcessor** * **processor**
- unsigned int **sampleRate**
- unsigned int **sampleSize**
- unsigned int **channels**

Campos

rAudioBuffer* buffer

unsigned int channels

rAudioProcessor* processor

unsigned int sampleRate

unsigned int sampleSize

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura AutomationEvent

`#include <raylib.h>`

Campos de Dados

- unsigned int **frame**
- unsigned int **type**
- int **params** [4]

Campos

unsigned int frame

int params[4]

unsigned int type

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura AutomationEventList

```
#include <raylib.h>
```

Campos de Dados

- unsigned int **capacity**
- unsigned int **count**
- **AutomationEvent** * **events**

Campos

unsigned int capacity

unsigned int count

AutomationEvent* events

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura BoneInfo

```
#include <raylib.h>
```

Campos de Dados

- char **name** [32]
- int **parent**

Campos

char name[32]

int parent

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura BoundingBox

```
#include <raylib.h>
```

Campos de Dados

- **Vector3 min**
- **Vector3 max**

Campos

Vector3 max

Vector3 min

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Camera2D

```
#include <raylib.h>
```

Campos de Dados

- **Vector2 offset**
- **Vector2 target**
- float **rotation**
- float **zoom**

Campos

Vector2 offset

float rotation

Vector2 target

float zoom

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Camera3D

```
#include <raylib.h>
```

Campos de Dados

- **Vector3 position**
- **Vector3 target**
- **Vector3 up**
- float **fovy**
- int **projection**

Campos

float fovy

Vector3 position

int projection

Vector3 target

Vector3 up

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Color

```
#include <raylib.h>
```

Campos de Dados

- unsigned char **r**
- unsigned char **g**
- unsigned char **b**
- unsigned char **a**

Campos

unsigned char a

unsigned char b

unsigned char g

unsigned char r

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Fila

```
#include <fila.h>
```

Campos de Dados

- int * **array**
- int **inicio**
- int **fim**
- int **tamanho**
- int **capacidade**

Campos

int* array

int capacidade

int fim

int inicio

int tamanho

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**fila.h**

Referência da Estrutura FilePathList

```
#include <raylib.h>
```

Campos de Dados

- unsigned int **capacity**
- unsigned int **count**
- char ** **paths**

Campos

unsigned int capacity

unsigned int count

char** paths

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Font

```
#include <raylib.h>
```

Campos de Dados

- int **baseSize**
- int **glyphCount**
- int **glyphPadding**
- **Texture2D texture**
- **Rectangle** * **recs**
- **GlyphInfo** * **glyphs**

Campos

int baseSize

int glyphCount

int glyphPadding

GlyphInfo* glyphs

Rectangle* recs

Texture2D texture

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura GlyphInfo

```
#include <raylib.h>
```

Campos de Dados

- int **value**
- int **offsetX**
- int **offsetY**
- int **advanceX**
- **Image image**

Campos

int advanceX

Image image

int offsetX

int offsetY

int value

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Image

```
#include <raylib.h>
```

Campos de Dados

- void * **data**
- int **width**
- int **height**
- int **mipmaps**
- int **format**

Campos

void* data

int format

int height

int mipmaps

int width

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Material

```
#include <raylib.h>
```

Campos de Dados

- **Shader shader**
- **MaterialMap** * **maps**
- float **params** [4]

Campos

MaterialMap* maps

float params[4]

Shader shader

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura MaterialMap

```
#include <raylib.h>
```

Campos de Dados

- **Texture2D texture**
- **Color color**
- float **value**

Campos

Color color

Texture2D texture

float value

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Matrix

```
#include <raylib.h>
```

Campos de Dados

- float **m0**
- float **m4**
- float **m8**
- float **m12**
- float **m1**
- float **m5**
- float **m9**
- float **m13**
- float **m2**
- float **m6**
- float **m10**
- float **m14**
- float **m3**
- float **m7**
- float **m11**
- float **m15**

Campos

float m0

float m1

float m10

float m11

float m12

float m13

float m14

float m15

float m2

float m3

float m4

float m5

float m6

float m7

float m8

float m9

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Mesh

```
#include <raylib.h>
```

Campos de Dados

- int **vertexCount**
- int **triangleCount**
- float * **vertices**
- float * **texcoords**
- float * **texcoords2**
- float * **normals**
- float * **tangents**
- unsigned char * **colors**
- unsigned short * **indices**
- float * **animVertices**
- float * **animNormals**
- unsigned char * **boneIds**
- float * **boneWeights**
- **Matrix** * **boneMatrices**
- int **boneCount**
- unsigned int **vaoId**
- unsigned int * **vboId**

Campos

float* animNormals

float* animVertices

int boneCount

unsigned char* boneIds

Matrix* boneMatrices

float* boneWeights

unsigned char* colors

unsigned short* indices

float* normals

float* tangents

float* texcoords

float* texcoords2

int triangleCount

unsigned int vaoId

unsigned int* vboId

int vertexCount

float* vertices

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Model

```
#include <raylib.h>
```

Campos de Dados

- **Matrix transform**
- int **meshCount**
- int **materialCount**
- **Mesh** * **meshes**
- **Material** * **materials**
- int * **meshMaterial**
- int **boneCount**
- **BoneInfo** * **bones**
- **Transform** * **bindPose**

Campos

Transform* bindPose

int boneCount

BoneInfo* bones

int materialCount

Material* materials

int meshCount

Mesh* meshes

int* meshMaterial

Matrix transform

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura ModelAnimation

```
#include <raylib.h>
```

Campos de Dados

- int **boneCount**
- int **frameCount**
- **BoneInfo** * **bones**
- **Transform** ** **framePoses**
- char **name** [32]

Campos

int boneCount

BoneInfo* bones

int frameCount

Transform** framePoses

char name[32]

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Music

```
#include <raylib.h>
```

Campos de Dados

- **AudioStream stream**
- unsigned int **frameCount**
- **bool looping**
- int **ctxType**
- void * **ctxData**

Campos

void* ctxData

int ctxType

unsigned int frameCount

bool looping

AudioStream stream

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura NPatchInfo

`#include <raylib.h>`

Campos de Dados

- **Rectangle source**
- int **left**
- int **top**
- int **right**
- int **bottom**
- int **layout**

Campos

int bottom

int layout

int left

int right

Rectangle source

int top

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Posicao

Campos de Dados

- int **x**
- int **y**

Campos

int x

int y

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**main.c**

Referência da Estrutura Ray

```
#include <raylib.h>
```

Campos de Dados

- **Vector3 position**
- **Vector3 direction**

Campos

Vector3 direction

Vector3 position

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura RayCollision

```
#include <raylib.h>
```

Campos de Dados

- **bool hit**
- float **distance**
- **Vector3 point**
- **Vector3 normal**

Campos

float distance

bool hit

Vector3 normal

Vector3 point

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Rectangle

```
#include <raylib.h>
```

Campos de Dados

- float **x**
- float **y**
- float **width**
- float **height**

Campos

float height

float width

float x

float y

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura RenderTexture

```
#include <raylib.h>
```

Campos de Dados

- unsigned int **id**
- **Texture texture**
- **Texture depth**

Campos

Texture depth

unsigned int id

Texture texture

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Shader

```
#include <raylib.h>
```

Campos de Dados

- unsigned int **id**
- int * **locs**

Campos

unsigned int id

int* locs

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Sound

```
#include <raylib.h>
```

Campos de Dados

- **AudioStream stream**
- unsigned int **frameCount**

Campos

unsigned int frameCount

AudioStream stream

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Stack

```
#include <pilha.h>
```

Campos de Dados

- int * **data**
- int **top**
- int **limit**

Campos

int* data

int limit

int top

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**pilha.h**

Referência da Estrutura Texture

```
#include <raylib.h>
```

Campos de Dados

- unsigned int **id**
- int **width**
- int **height**
- int **mipmaps**
- int **format**

Campos

int format

int height

unsigned int id

int mipmaps

int width

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Transform

```
#include <raylib.h>
```

Campos de Dados

- **Vector3 translation**
- **Quaternion rotation**
- **Vector3 scale**

Campos

Quaternion rotation

Vector3 scale

Vector3 translation

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Vector2

```
#include <raylib.h>
```

Campos de Dados

- float **x**
- float **y**

Campos

float x

float y

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Vector3

```
#include <raylib.h>
```

Campos de Dados

- float **x**
- float **y**
- float **z**

Campos

float x

float y

float z

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Vector4

```
#include <raylib.h>
```

Campos de Dados

- float **x**
- float **y**
- float **z**
- float **w**

Campos

float w

float x

float y

float z

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura VrDeviceInfo

`#include <raylib.h>`

Campos de Dados

- int **hResolution**
- int **vResolution**
- float **hScreenSize**
- float **vScreenSize**
- float **eyeToScreenDistance**
- float **lensSeparationDistance**
- float **interpupillaryDistance**
- float **lensDistortionValues** [4]
- float **chromaAbCorrection** [4]

Campos

float chromaAbCorrection[4]

float eyeToScreenDistance

int hResolution

float hScreenSize

float interpupillaryDistance

float lensDistortionValues[4]

float lensSeparationDistance

int vResolution

float vScreenSize

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura VrStereoConfig

```
#include <raylib.h>
```

Campos de Dados

- **Matrix projection** [2]
- **Matrix viewOffset** [2]
- float **leftLensCenter** [2]
- float **rightLensCenter** [2]
- float **leftScreenCenter** [2]
- float **rightScreenCenter** [2]
- float **scale** [2]
- float **scaleIn** [2]

Campos

float leftLensCenter[2]

float leftScreenCenter[2]

Matrix projection[2]

float rightLensCenter[2]

float rightScreenCenter[2]

float scale[2]

float scaleIn[2]

Matrix viewOffset[2]

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Referência da Estrutura Wave

```
#include <raylib.h>
```

Campos de Dados

- unsigned int **frameCount**
- unsigned int **sampleRate**
- unsigned int **sampleSize**
- unsigned int **channels**
- void * **data**

Campos

unsigned int channels

void* data

unsigned int frameCount

unsigned int sampleRate

unsigned int sampleSize

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/ASUS/Documents/GitHub/trabalhoED/**raylib.h**

Arquivos

Referência do Arquivo C:/Users/ASUS/Documents/GitHub/trabalhoED/fila.c

```
#include <stdlib.h>
#include "fila.h"
```

Funções

- **Fila** * **InitializeQueue** (int capacidade)
- int **Enqueue** (**Fila** *fila, int numero)
- int **Dequeue** (**Fila** *fila)
- int **IsQueueEmpty** (**Fila** *fila)
- int **IsQueueFull** (**Fila** *fila)
- void **DestroyQueue** (**Fila** *fila)

Funções

int Dequeue (Fila * fila)

```
                        {

    if(IsQueueEmpty(fila))
        return '\0';

    int numero = fila->array[fila->inicio];
    fila->tamanho--;
    if(fila->inicio != fila->fim){
        if(fila->inicio == fila->capacidade - 1)
            fila->inicio = 0;
        else
            fila->inicio++;
    }
    return numero;
}
```

void DestroyQueue (Fila * fila)

```
                              {
    free (fila->array);
    free (fila);
}
```

int Enqueue (Fila * fila, int numero)

```
                                    {

    if(IsQueueFull(fila))
        return 0;

    if(!(IsQueueEmpty(fila))){
        if(fila->fim == fila->capacidade - 1)
            fila->fim = 0;
        else
            fila->fim++;
    }

    fila->array[fila->fim] = numero;
    fila->tamanho++;
    return 1;
}
```

Fila * InitializeQueue (int capacidade)

```
                                      {

    Fila *fila = (Fila*)malloc(sizeof(Fila));
    fila->array = (int*)malloc(capacidade*sizeof(int));

    fila->inicio = 0;
     fila->fim = 0;
     fila->tamanho = 0;
     fila->capacidade = capacidade;

     return fila;
 }
```

int IsQueueEmpty (Fila * fila)

```
                             {
     return fila->tamanho == 0;
 }
```

int IsQueueFull (Fila * fila)

```
                            {
     return fila->tamanho == fila->capacidade;
 }
```

Referência do Arquivo C:/Users/ASUS/Documents/GitHub/trabalhoED/fila.h

Estruturas de Dados

struct FilaFunções

- **Fila** * **InitializeQueue** (int capacidade)
- int **Enqueue** (**Fila** *fila, int numero)
- int **Dequeue** (**Fila** *fila)
- int **IsQueueEmpty** (**Fila** *fila)
- int **IsQueueFull** (**Fila** *fila)
- void **DestroyQueue** (**Fila** *fila)

Funções

int Dequeue (Fila * fila)

```
                        {

    if(IsQueueEmpty(fila))
        return '\0';

    int numero = fila->array[fila->inicio];
    fila->tamanho--;
    if(fila->inicio != fila->fim){
        if(fila->inicio == fila->capacidade - 1)
            fila->inicio = 0;
        else
            fila->inicio++;
    }
    return numero;
}
```

void DestroyQueue (Fila * fila)

```
                              {
    free (fila->array);
    free (fila);
}
```

int Enqueue (Fila * fila, int numero)

```
                                    {

    if(IsQueueFull(fila))
        return 0;

    if(!(IsQueueEmpty(fila))){
        if(fila->fim == fila->capacidade - 1)
            fila->fim = 0;
        else
            fila->fim++;
    }

    fila->array[fila->fim] = numero;
    fila->tamanho++;
    return 1;
}
```

Fila * InitializeQueue (int capacidade)

```
                                    {

    Fila *fila = (Fila*)malloc(sizeof(Fila));
    fila->array = (int*)malloc(capacidade*sizeof(int));

    fila->inicio = 0;
     fila->fim = 0;
     fila->tamanho = 0;
     fila->capacidade = capacidade;

     return fila;
}
```

int IsQueueEmpty (Fila * fila)

```
                            {
     return fila->tamanho == 0;
}
```

int IsQueueFull (Fila * fila)

```
                           {
     return fila->tamanho == fila->capacidade;
}
```

fila.h

Ir para a documentação desse arquivo.

```
#ifndef FILA_H
#define FILA_H
typedef struct {

    int *array;
    int inicio;
    int fim;
    int tamanho;
    int capacidade;

} Fila;

Fila *InitializeQueue(int capacidade);
int Enqueue(Fila *fila, int numero);
int Dequeue(Fila *fila);
int IsQueueEmpty(Fila *fila);
int IsQueueFull(Fila *fila);
void DestroyQueue(Fila *fila);

#endif
```

Referência do Arquivo C:/Users/ASUS/Documents/GitHub/trabalhoED/main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "raylib.h"
#include "pilha.h"
#include "fila.h"
```

Estruturas de Dados

struct PosicaoDefinições e Macros

- #define **VELOCIDADE**   3

Definições de Tipos

- typedef struct Posicao **Posicao**

Funções

- void **DesenharLabirinto** (int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], **Posicao** atual, int destino[2], int desenhaDFS, int desenhaBFS, int CELL_SIZE)
- void **DesenharLabirintoFinal** (int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], int destino[2], int CELL_SIZE)
- int **DFS** (int x, int y, int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], int destino[2], **Stack** *pilha, int CELL_SIZE)
- int **BFS** (int x, int y, int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], int destino[2], **Fila** *fila, int CELL_SIZE)
- int **main** ()

Definições e macros

#define VELOCIDADE   3

Definições dos tipos

typedef struct Posicao Posicao

Funções

int BFS (int x, int y, int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], int destino[2], Fila * fila, int CELL_SIZE)

```
{

    if(labirinto[x][y] == 1){ // se comecar numa parede, falhou
        return 0;
    }
    int pos = (x*TAM) + y; // posicao eh codifificada
    Enqueue(fila, pos);     // enfileira a posicao
    visitadoBFS[x][y] = 1;   //  primeiro vertice eh visitado


    int dx[] = {-1, 1, 0, 0}; //anda para cima, anda para baixo, anda para esquerda e anda para a direita
    int dy[] = {0, 0, -1, 1};


    while(!IsQueueEmpty(fila) && !WindowShouldClose()){ // enquanto a fila nao esta vazia

        int pos = Dequeue(fila);    /*desenfileira a primeira posicao da fila*/
        Posicao atual;  // declara a struct posicao

        atual.x = pos / TAM;  // decodifica o x
        atual.y = pos % TAM;  // decodifica o y

        WaitTime(0.3); // tempo de espera para agir
        DesenharLabirinto(TAM, labirinto, visitadoDFS, visitadoBFS, atual, destino, 0, 1, CELL_SIZE); // desenha o labirinto atual

        if (atual.x == destino[0] && atual.y == destino[1])
            return 1;  // destino encontrado


        for(int i = 0; i < 4; i++){ // observa as 4 posicoes
            int nx = atual.x + dx[i]; // olha para cima, baixo, esquerda, e direita
            int ny = atual.y + dy[i];

            int pos_fake;   // posicao observada
            if(nx >= 0 && ny >= 0 && nx < TAM && ny < TAM && labirinto[nx][ny] == 0 && !visitadoBFS[nx][ny]){ // caso a posicao observada preencha os requisitos, ela eh uma pos valida
                pos_fake = (nx * TAM) + ny; // codifica o x e o y para a pos observaa
                Enqueue(fila, pos_fake);   // empilha a pos observada
                visitadoBFS[nx][ny] = 1; // marca o vertice x y como visita
            }
        }
    }

    return 0; // caso saia do while e nao ache, nao ha caminho
}
```

void DesenharLabirinto (int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], Posicao atual, int destino[2], int desenhaDFS, int desenhaBFS, int CELL_SIZE)

```
{
    // inicia o processo de desenho na tela
    BeginDrawing();

    // limpa a tela com a cor branca (constante RAYWHITE da Raylib)
    ClearBackground(RAYWHITE);

    //if(atual.x != destino[0] || atual.y != destino[1]){
        // desenha um quadrado vermelho no destino (para o painel DFS)
        // a posição é ajustada somando 3 às coordenadas de destino e multiplicando pelo tamanho da célula
        DrawRectangle((destino[1]+3)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, RED);

        // desenha um quadrado vermelho no destino (para o painel BFS)
        // aqui a posição horizontal é deslocada em (destino+9+TAM) células para separar os dois painéis
        DrawRectangle((destino[1]+9+TAM)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, RED);
    //}




    // loop para desenhar as células do labirinto e suas bordas extras
    // o loop vai de -1 até TAM (inclusive) para criar uma "moldura" em volta do labirinto
    for (int i = -1; i < TAM+1; i++){
        for (int j = -1; j < TAM+1; j++){
            // se a célula é uma parede (valor 1 na matriz do labirinto)
            if(i >= 0 && i < TAM && j >=0 && j < TAM){
                if(labirinto[i][j] == 1){
                    // desenha a parede no painel DFS com cor cinza (GRAY)
                    DrawRectangle((j+3) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, GRAY);
                    // desenha a parede no painel BFS; nota o deslocamento horizontal para separar os painéis
                    DrawRectangle((j+TAM+9) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, GRAY);
                }
            }
            // se a célula faz parte da moldura externa (bordas do labirinto)
            if( i == -1 || i == TAM || j == -1|| j == TAM){
                // desenha a borda preta no painel DFS
                DrawRectangle((j+3) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, BLACK);
                // desenha a borda preta no painel BFS
                DrawRectangle((j+TAM+9) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, BLACK);
            }
        }
    }

    // desenha as linhas da grade sobre todo o espaço dos dois painéis
    // o loop percorre todas as linhas necessárias considerando o tamanho total (TAM+6)*2
    for (int i = 0; i <= (TAM+6)*2; i++){
        // linha horizontal
        DrawLine(0, i*CELL_SIZE, ((TAM+6)*CELL_SIZE)*2, i*CELL_SIZE, BLACK);
        // linha vertical
        DrawLine(i * CELL_SIZE, 0, i*CELL_SIZE, ((TAM+6)*CELL_SIZE)*2, BLACK);
    }

    int xBFS;
    int y;
    // desenha as células internas do labirinto para cada painel (DFS e BFS)
    for (int i = 0; i < TAM; i++){
        for (int j = 0; j < TAM; j++){
            // calcula a posição X para o painel DFS (deslocado 3 células à direita)
            int xDFS = (3+j)*CELL_SIZE;
            // calcula a posição Y comum para ambos os painéis (deslocado 3 células para baixo)
            int y = (3+i)*CELL_SIZE;
            // calcula a posição X para o painel BFS (deslocado para a direita: 9+TAM células)
            int xBFS = (9+TAM+j)*CELL_SIZE;

            // variável calculada, mas que não é utilizada nas próximas linhas
            int posDestino = (destino[0]*TAM)+destino[1];

            // se a célula atual é a posição "atual" (passada como parâmetro)
            if(i == atual.x && j == atual.y ){
                // se o DFS visitou essa célula
                if(visitadoDFS[i][j] == 1){
                    // se o flag de desenho do DFS está ativado, pinta a célula de verde
                    if(desenhaDFS == 1){
                        DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, GREEN);
                        // desenha a borda da célula
                        DrawRectangleLines(xDFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                    }else{
                        // se não, pinta a célula de magenta
                        DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, MAGENTA);
                        DrawRectangleLines(xDFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                    }
                }
                // se o BFS visitou essa célula
                if(visitadoBFS[i][j] == 1){
                    // se o flag de desenho do BFS está ativado, pinta a célula de verde
                    if(desenhaBFS == 1){
                        DrawRectangle(xBFS, y, CELL_SIZE, CELL_SIZE, GREEN);
                        DrawRectangleLines(xBFS, y, CELL_SIZE, CELL_SIZE, BLACK);
```

```c
                    }else{
                        // caso contrário, pinta a célula de roxo escuro (SKYBLUE)
                        DrawRectangle(xBFS, y, CELL_SIZE, CELL_SIZE, SKYBLUE);
                        DrawRectangleLines(xBFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                    }
                }
            }
            // caso a célula atual não seja o destino (ou seja, não é a célula de chegada)
            else if(i != destino[0] || j != destino[1]){
                // para o painel DFS:
                // se a célula foi visitada, pinta de magenta
                if(visitadoDFS[i][j] == 1){
                    DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, MAGENTA);
                    DrawRectangleLines(xDFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                }else{
                    // se não foi visitada, pinta de branco (BLANK) com borda preta
                    DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, BLANK);
                    DrawRectangleLines(xDFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                }

                // para o painel BFS:
                // se a célula foi visitada, pinta de roxo escuro
                if(visitadoBFS[i][j] == 1){
                    DrawRectangle(xBFS, y, CELL_SIZE, CELL_SIZE, SKYBLUE);
                    DrawRectangleLines(xBFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                }else{
                    // se não foi visitada, pinta de branco; nota o pequeno ajuste no deslocamento horizontal (pode ser para manter alinhamento)
                    DrawRectangle(xBFS+TAM+9, y, CELL_SIZE, CELL_SIZE, BLANK);
                    DrawRectangleLines(xBFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                }
            }else{
                if(visitadoDFS[i][j] == 1){
                    DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, GREEN);
                    DrawRectangleLines(xDFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                }

                if(visitadoBFS[i][j] == 1){
                    DrawRectangle(xBFS, y, CELL_SIZE, CELL_SIZE, GREEN);
                    DrawRectangleLines(xBFS, y, CELL_SIZE, CELL_SIZE, BLACK);
                }
            }
        }
    }

    // redesenha os quadrados de destino para garantir que fiquem visíveis por cima de outros elementos
    // DrawRectangle((destino[1]+3)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, RED);
    // DrawRectangle((destino[1]+9+TAM)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, RED);

    // Desenha os rótulos "DFS" e "BFS" acima dos labirintos
    int fontSize = CELL_SIZE;
    const char *dfsText = "DFS";
    const char *bfsText = "BFS";

    // Calcula posição para "DFS"
    int dfsTextWidth = MeasureText(dfsText, fontSize);
    int dfsX = 3 * CELL_SIZE + (TAM * CELL_SIZE) / 2 - dfsTextWidth / 2;
    int dfsY = 1 * CELL_SIZE;
    DrawText(dfsText, dfsX, dfsY, fontSize, MAGENTA);

    // Calcula posição para "BFS"
    int bfsTextWidth = MeasureText(bfsText, fontSize);
    int bfsX = (TAM + 9) * CELL_SIZE + (TAM * CELL_SIZE) / 2 - bfsTextWidth / 2;
    int bfsY = 1 * CELL_SIZE;
    DrawText(bfsText, bfsX, bfsY, fontSize, SKYBLUE);


    EndDrawing();
}
```

void DesenharLabirintoFinal (int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], int destino[2], int CELL_SIZE)

```
{
    BeginDrawing();
    ClearBackground(RAYWHITE);

    // Desenha os quadrados de destino para ambos os painéis
    DrawRectangle((destino[1]+3)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, RED);
    DrawRectangle((destino[1]+9+TAM)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, RED);

    // Desenha paredes e bordas (moldura) sem acessar índices inválidos
    for (int i = -1; i < TAM+1; i++){
        for (int j = -1; j < TAM+1; j++){
            if(i >= 0 && i < TAM && j >= 0 && j < TAM){
                if(labirinto[i][j] == 1){
                    // Painel DFS
                    DrawRectangle((j+3) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, GRAY);
                    // Painel BFS
                    DrawRectangle((j+TAM+9) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, GRAY);
                }
            }
            // Desenha as bordas da moldura
            if(i == -1 || i == TAM || j == -1 || j == TAM){
                DrawRectangle((j+3) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, BLACK);
                DrawRectangle((j+TAM+9) * CELL_SIZE, (i+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, BLACK);
            }
        }
    }

    // Desenha a grade sobre ambos os painéis
    for (int i = 0; i <= (TAM+6)*2; i++){
        DrawLine(0, i*CELL_SIZE, ((TAM+6)*CELL_SIZE)*2, i*CELL_SIZE, BLACK);
        DrawLine(i * CELL_SIZE, 0, i*CELL_SIZE, ((TAM+6)*CELL_SIZE)*2, BLACK);
    }

    int xDFS;
    int y;
    int xBFS;
    // Desenha as células internas para cada painel
    for (int i = 0; i < TAM; i++){
        for (int j = 0; j < TAM; j++){
            xDFS = (3+j)*CELL_SIZE;       // Posição X para o painel DFS
            y = (3+i)*CELL_SIZE;           // Posição Y comum
            xBFS = (9+TAM+j)*CELL_SIZE;      // Posição X para o painel BFS

            // Painel DFS: se visitado, pinta de magenta; senão, branco
            if(visitadoDFS[i][j] == 1)
                DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, MAGENTA);
            else
                DrawRectangle(xDFS, y, CELL_SIZE, CELL_SIZE, BLANK);
            DrawRectangleLines(xDFS, y, CELL_SIZE, CELL_SIZE, BLACK);

            // Painel BFS: se visitado, pinta de roxo escuro; senão, branco
            if(visitadoBFS[i][j] == 1)
                DrawRectangle(xBFS, y, CELL_SIZE, CELL_SIZE, SKYBLUE);
            else
                DrawRectangle(xBFS, y, CELL_SIZE, CELL_SIZE, BLANK);
            DrawRectangleLines(xBFS, y, CELL_SIZE, CELL_SIZE, BLACK);
        }
    }


    //caso o caminho seja encontrado, a casa destino fica verde
    if(visitadoDFS[destino[0]][destino[1]] == 1){
        DrawRectangle((destino[1]+3)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, GREEN);

    }
    if(visitadoBFS[destino[0]][destino[1]] == 1){
        DrawRectangle((destino[1]+9+TAM)*CELL_SIZE, (destino[0]+3)*CELL_SIZE, CELL_SIZE, CELL_SIZE, GREEN);
    }

    //desenha as linhas
    for (int i = 0; i <= (TAM+6)*2; i++){
        DrawLine(0, i*CELL_SIZE, ((TAM+6)*CELL_SIZE)*2, i*CELL_SIZE, BLACK);
        DrawLine(i * CELL_SIZE, 0, i*CELL_SIZE, ((TAM+6)*CELL_SIZE)*2, BLACK);
    }

    // Desenha os rótulos "DFS" e "BFS" acima dos labirintos
    int fontSize = CELL_SIZE;
    const char *dfsText = "DFS";
    const char *bfsText = "BFS";
    const char *NEText  = "Caminho Não Encontrado";

    // Calcula posição para "DFS"
    int dfsTextWidth = MeasureText(dfsText, fontSize);
    int dfsX = 3 * CELL_SIZE + (TAM * CELL_SIZE) / 2 - dfsTextWidth / 2;
    int dfsY = 1 * CELL_SIZE;
    DrawText(dfsText, dfsX, dfsY, fontSize, MAGENTA);

    // Calcula posição para "BFS"
    int bfsTextWidth = MeasureText(bfsText, fontSize);
    int bfsX = (TAM + 9) * CELL_SIZE + (TAM * CELL_SIZE) / 2 - bfsTextWidth / 2;
    int bfsY = 1 * CELL_SIZE;
    DrawText(bfsText, bfsX, bfsY, fontSize, SKYBLUE);

    if(visitadoDFS[destino[0]][destino[1]] != 1){
        int NETextWidth = MeasureText(NEText, fontSize);
        int NEX = NETextWidth/2;
        int NEY = (TAM + 5) * CELL_SIZE;
        DrawText(NEText, NEX, NEY, fontSize, BLACK);
    }


    EndDrawing();
}
```

int DFS (int x, int y, int TAM, int labirinto[TAM][TAM], int visitadoDFS[TAM][TAM], int visitadoBFS[TAM][TAM], int destino[2], Stack * pilha, int CELL_SIZE)

```
{

    if(labirinto[x][y] == 1) // se comecar numa parede, falhou
        return 0;

    int pos = (x*TAM) + y;  // posicao eh codifificada
    Push(pilha, pos);   // empilha a posicao
     visitadoDFS[x][y] = 1;  //  primeiro vertice eh visitado


     while (!IsStackEmpty(pilha) && !WindowShouldClose()) {   // enquanto a pilha nao estiver vazia
         int pos = Peek(pilha);   // recebe o topo da pilha como posicao a ser analisada
         int cx = pos / TAM;  // decodifica o x
         int cy = pos % TAM;  // decodifica o y
         WaitTime(0.3); // tempo de espera para agir
         DesenharLabirinto(TAM, labirinto, visitadoDFS, visitadoBFS, (Posicao){.x = cx, .y = cy}, destino, 1, 0, CELL_SIZE);   // desenha toda vez que comeca a buscar um novo vertice

         if (cx == destino[0] && cy == destino[1]) return 1;  // destino encontrado

         int vizinhoEncontrado = 0; // variavel utilizada para definir se há vizinho disponivel
         int dx[] = {0, 1, 0, -1}; // anda para direita, anda para baixo, anda para esquerda, anda para cima
         int dy[] = {1, 0, -1, 0};

         for (int i = 0; i < 4; i++) { // inicia as 4 movimentacoes
```

```c
            int nx = cx + dx[i]; // x observado recebe seu x mais seu movimento
            int ny = cy + dy[i]; // y observado recebe seu y mais seu movimento
            if (nx >= 0 && nx < TAM && ny >= 0 && ny < TAM && labirinto[nx][ny] == 0 && !visitadoDFS[nx][ny]) { // primeiro verifica se nao esta olhando para fora do labirinto, apos isso, verifica se a posicao olhada eh uma area disponivel, por fim, verifica se a posicao ja nao foi visitada
                Push(pilha, nx * TAM + ny); // empilha a recem descoberta posicao
                visitadoDFS[nx][ny] = 1;    // marca a posicao descoberta como visitada
                vizinhoEncontrado = 1; //altera a variavel pois há vizinho disponivel
                break;
            }
        }

        //caso nao haja vizinho disponivel, retorna para o "pixel" anterior
        if(!vizinhoEncontrado){
            Pop(pilha);
        }

    }
    return 0;  // caso saia do while sem achar, nao a caminho.
}
```

int main ()

```c
          {

    int TAM, inicioX, inicioY, destino[2];
    printf("Insira o tamanho do seu labirinto quadrado: \n");
    scanf("%d", &TAM); // inicialização das variáveis
    int labirinto[TAM][TAM]; // matriz que define o labirinto
    int visitadoDFS[TAM][TAM];  // matriz das coordenadas visitas pelo dfs
    int visitadoBFS[TAM][TAM];  // matriz das coordenadas visitas pelo bfs
    printf("Insira o labirinto desejado para realizar as buscas (0 para possíveis caminhos, 1 para paredes): \n");
    for (int i = 0; i < TAM; i++){
        for (int j = 0; j < TAM; j++)
        {
            scanf("%d", &labirinto[i][j]);
            visitadoBFS[i][j] = 0;
            visitadoDFS[i][j] = 0;
        }
    }
    printf("Insira o local (comprimento e altura) desejado para se iniciar as buscas (0 a %d): \n", TAM-1);
    scanf("%d %d", &inicioX, &inicioY);
    printf("Insira o local (comprimento e altura) desejado para se terminar as buscas (0 a %d): \n", TAM-1);
    scanf("%d %d", &destino[0], &destino[1]);
    int CELL_SIZE = 300/TAM; // o tamanho de cada "pixel" é equivalente a 300 dividido pelo comprimento/largura do labirinto








    Stack *pilha = InitializeStack(TAM*TAM);   // inicializa a pilha com tamanho 25

    Fila *fila = InitializeQueue(TAM*TAM);    // inicializa a fila com tamanho 25

    Posicao atual = {0};    // posicao inicial eh a (0,0)






    InitWindow(((TAM+6)*CELL_SIZE)*2, (TAM+6)*CELL_SIZE, "DFS vs BFS"); // inicializacao da janela grafica
    //altura: (TAM+6)*CELL_SIZE
    // largura: ((TAM+6)*CELL_SIZE)*2 — a largura da janela (ajusta para mostrar os dois algoritmos lado a lado)
    // "DFS vs BFS": Título da janela

    SetTargetFPS(60);   // define a quantos frames per second vai rodar a animação


    int resultadoDFS = -1;
    while (!WindowShouldClose()){ // mantem a janela rodando enquanto o usuario nao quer fechar.
        if(resultadoDFS == -1){ // so sera executado no primeiro loop
            WaitTime(1);    // espera 1 segundo para executar
            resultadoDFS = DFS(inicioX, inicioY, TAM, labirinto, visitadoDFS, visitadoBFS, destino, pilha, CELL_SIZE); // caso return == 0 ,nao ha caminho
            if(!resultadoDFS)  printf("Caminho nao encontrado"); // caso falhe o caminho dfs, printa que nao achou
            if(!WindowShouldClose()){
                if(!BFS(inicioX, inicioY, TAM, labirinto, visitadoDFS, visitadoBFS, destino, fila, CELL_SIZE)) printf("Caminho nao encontrado\n"); // mesma logica do anterior
            }
        }else{
            DesenharLabirintoFinal(TAM, labirinto, visitadoDFS, visitadoBFS, destino, CELL_SIZE); // desenha o labirinto final

        }
    }
    CloseWindow(); //funcao fecha de fato a janela


    DestroyStack(pilha);    // destroi a pilha
    DestroyQueue(fila);      //destroi a fila
}
```

Referência do Arquivo C:/Users/ASUS/Documents/GitHub/trabalhoED/pilha.c

```
#include <stdlib.h>
#include "pilha.h"
```

Funções

- **Stack** * **InitializeStack** (int limit)
- int **IsStackEmpty** (**Stack** *s)
- int **IsStackFull** (**Stack** *s)
- void **Push** (**Stack** *s, int value)
- int **Pop** (**Stack** *s)
- int **Peek** (**Stack** *s)
- void **DestroyStack** (**Stack** *s)

Funções

void DestroyStack (Stack * s)

```
                             {
    free(s->data);
    s->data = NULL;
    s->top = -1;
    s->limit = 0;
}
```

Stack * InitializeStack (int limit)

```
                                {
    Stack *s = (Stack*)malloc(sizeof(Stack));
    s->data = (int*)malloc(limit *sizeof(int));
    s->top = -1;
    s->limit = limit;
}
```

int IsStackEmpty (Stack * s)

```
                             {
    return s->top == -1;
}
```

int IsStackFull (Stack * s)

```
                            {
    return s->top == s->limit-1;
}
```

int Peek (Stack * s)

```
                     {
    if(IsStackEmpty(s)) return -1;
    return s->data[s->top];
}
```

int Pop (Stack * s)

```
                    {
    if(IsStackEmpty(s)) return -1;
    s->top--;
    return s->data[s->top+1];
}
```

void Push (Stack * s, int value)

```
                                {
    if(IsStackFull(s)) return;
    s->data[++s->top] = value;
}
```

Referência do Arquivo C:/Users/ASUS/Documents/GitHub/trabalhoED/pilha.h

Estruturas de Dados

struct StackFunções

- **Stack** * **InitializeStack** (int limit)
- int **IsStackEmpty** (**Stack** *s)
- int **IsStackFull** (**Stack** *s)
- void **Push** (**Stack** *s, int value)
- int **Pop** (**Stack** *s)
- int **Peek** (**Stack** *s)
- void **DestroyStack** (**Stack** *s)

Funções

void DestroyStack (Stack * s)

```
                             {
       free(s->data);
       s->data = NULL;
       s->top = -1;
       s->limit = 0;
   }
```

Stack * InitializeStack (int limit)

```
                                   {
      Stack *s = (Stack*)malloc(sizeof(Stack));
      s->data = (int*)malloc(limit *sizeof(int));
      s->top = -1;
      s->limit = limit;
  }
```

int IsStackEmpty (Stack * s)

```
                             {
       return s->top == -1;
   }
```

int IsStackFull (Stack * s)

```
                            {
       return s->top == s->limit-1;
   }
```

int Peek (Stack * s)

```
                     {
       if(IsStackEmpty(s)) return -1;
       return s->data[s->top];
   }
```

int Pop (Stack * s)

```
                    {
```

```
    if(IsStackEmpty(s)) return -1;
    s->top--;
    return s->data[s->top+1];
}
```

void Push (Stack * s, int value)

```
                                {
    if(IsStackFull(s)) return;
    s->data[++s->top] = value;
}
```

pilha.h

Ir para a documentação desse arquivo.

```
#ifndef PILHA_H
#define PILHA_H

typedef struct {
    int *data;
    int top;
    int limit;
} Stack;

Stack *InitializeStack( int limit );
int IsStackEmpty( Stack *s );
int IsStackFull( Stack *s );
void Push( Stack *s, int value );
int Pop( Stack *s );
int Peek( Stack *s );
void DestroyStack( Stack *s);
#endif
```

Referência do Arquivo C:/Users/ASUS/Documents/GitHub/trabalhoED/raylib.h

```
#include <stdarg.h>
```

Estruturas de Dados

- struct **Vector2**struct **Vector3**
- struct **Vector4**
- struct **Matrix**
- struct **Color**
- struct **Rectangle**
- struct **Image**
- struct **Texture**
- struct **RenderTexture**
- struct **NPatchInfo**
- struct **GlyphInfo**
- struct **Font**
- struct **Camera3D**
- struct **Camera2D**
- struct **Mesh**
- struct **Shader**
- struct **MaterialMap**
- struct **Material**
- struct **Transform**
- struct **BoneInfo**
- struct **Model**
- struct **ModelAnimation**
- struct **Ray**
- struct **RayCollision**
- struct **BoundingBox**
- struct **Wave**
- struct **AudioStream**
- struct **Sound**
- struct **Music**
- struct **VrDeviceInfo**
- struct **VrStereoConfig**
- struct **FilePathList**
- struct **AutomationEvent**
- struct **AutomationEventList**

Definições e Macros

- #define **RAYLIB_VERSION_MAJOR** 5
- #define **RAYLIB_VERSION_MINOR** 5
- #define **RAYLIB_VERSION_PATCH** 0
- #define **RAYLIB_VERSION** "5.5"
- #define **RLAPI**
- #define **PI** 3.14159265358979323846f
- #define **DEG2RAD** (**PI**/180.0f)
- #define **RAD2DEG** (180.0f/**PI**)
- #define **RL_MALLOC**(sz)
- #define **RL_CALLOC**(n, sz)
- #define **RL_REALLOC**(ptr, sz)
- #define **RL_FREE**(ptr)
- #define **CLITERAL**(type)
- #define **RL_COLOR_TYPE**
- #define **RL_RECTANGLE_TYPE**
- #define **RL_VECTOR2_TYPE**
- #define **RL_VECTOR3_TYPE**

- #define **RL_VECTOR4_TYPE**
- #define **RL_QUATERNION_TYPE**
- #define **RL_MATRIX_TYPE**
- #define **LIGHTGRAY CLITERAL**(**Color**){ 200, 200, 200, 255 }
- #define **GRAY CLITERAL**(**Color**){ 130, 130, 130, 255 }
- #define **DARKGRAY CLITERAL**(**Color**){ 80, 80, 80, 255 }
- #define **YELLOW CLITERAL**(**Color**){ 253, 249, 0, 255 }
- #define **GOLD CLITERAL**(**Color**){ 255, 203, 0, 255 }
- #define **ORANGE CLITERAL**(**Color**){ 255, 161, 0, 255 }
- #define **PINK CLITERAL**(**Color**){ 255, 109, 194, 255 }
- #define **RED CLITERAL**(**Color**){ 230, 41, 55, 255 }
- #define **MAROON CLITERAL**(**Color**){ 190, 33, 55, 255 }
- #define **GREEN CLITERAL**(**Color**){ 0, 228, 48, 255 }
- #define **LIME CLITERAL**(**Color**){ 0, 158, 47, 255 }
- #define **DARKGREEN CLITERAL**(**Color**){ 0, 117, 44, 255 }
- #define **SKYBLUE CLITERAL**(**Color**){ 102, 191, 255, 255 }
- #define **BLUE CLITERAL**(**Color**){ 0, 121, 241, 255 }
- #define **DARKBLUE CLITERAL**(**Color**){ 0, 82, 172, 255 }
- #define **PURPLE CLITERAL**(**Color**){ 200, 122, 255, 255 }
- #define **VIOLET CLITERAL**(**Color**){ 135, 60, 190, 255 }
- #define **DARKPURPLE CLITERAL**(**Color**){ 112, 31, 126, 255 }
- #define **BEIGE CLITERAL**(**Color**){ 211, 176, 131, 255 }
- #define **BROWN CLITERAL**(**Color**){ 127, 106, 79, 255 }
- #define **DARKBROWN CLITERAL**(**Color**){ 76, 63, 47, 255 }
- #define **WHITE CLITERAL**(**Color**){ 255, 255, 255, 255 }
- #define **BLACK CLITERAL**(**Color**){ 0, 0, 0, 255 }
- #define **BLANK CLITERAL**(**Color**){ 0, 0, 0, 0 }
- #define **MAGENTA CLITERAL**(**Color**){ 255, 0, 255, 255 }
- #define **RAYWHITE CLITERAL**(**Color**){ 245, 245, 245, 255 }
- #define **RL_BOOL_TYPE**
- #define **MOUSE_LEFT_BUTTON MOUSE_BUTTON_LEFT**
- #define **MOUSE_RIGHT_BUTTON MOUSE_BUTTON_RIGHT**
- #define **MOUSE_MIDDLE_BUTTON MOUSE_BUTTON_MIDDLE**
- #define **MATERIAL_MAP_DIFFUSE MATERIAL_MAP_ALBEDO**
- #define **MATERIAL_MAP_SPECULAR MATERIAL_MAP_METALNESS**
- #define **SHADER_LOC_MAP_DIFFUSE SHADER_LOC_MAP_ALBEDO**
- #define **SHADER_LOC_MAP_SPECULAR SHADER_LOC_MAP_METALNESS**
- #define **GetMouseRay GetScreenToWorldRay**

Definições de Tipos

- typedef enum **bool bool**
- typedef struct Vector2 **Vector2**
- typedef struct Vector3 **Vector3**
- typedef struct Vector4 **Vector4**
- typedef **Vector4 Quaternion**
- typedef struct Matrix **Matrix**
- typedef struct Color **Color**
- typedef struct Rectangle **Rectangle**
- typedef struct Image **Image**
- typedef struct Texture **Texture**
- typedef **Texture Texture2D**
- typedef **Texture TextureCubemap**
- typedef struct RenderTexture **RenderTexture**
- typedef **RenderTexture RenderTexture2D**
- typedef struct NPatchInfo **NPatchInfo**
- typedef struct GlyphInfo **GlyphInfo**
- typedef struct Font **Font**
- typedef struct Camera3D **Camera3D**
- typedef **Camera3D Camera**
- typedef struct Camera2D **Camera2D**

- typedef struct Mesh **Mesh**
- typedef struct Shader **Shader**
- typedef struct MaterialMap **MaterialMap**
- typedef struct Material **Material**
- typedef struct Transform **Transform**
- typedef struct BoneInfo **BoneInfo**
- typedef struct Model **Model**
- typedef struct ModelAnimation **ModelAnimation**
- typedef struct Ray **Ray**
- typedef struct RayCollision **RayCollision**
- typedef struct BoundingBox **BoundingBox**
- typedef struct Wave **Wave**
- typedef struct **rAudioBuffer rAudioBuffer**
- typedef struct **rAudioProcessor rAudioProcessor**
- typedef struct AudioStream **AudioStream**
- typedef struct Sound **Sound**
- typedef struct Music **Music**
- typedef struct VrDeviceInfo **VrDeviceInfo**
- typedef struct VrStereoConfig **VrStereoConfig**
- typedef struct FilePathList **FilePathList**
- typedef struct AutomationEvent **AutomationEvent**
- typedef struct AutomationEventList **AutomationEventList**
- typedef void(* **TraceLogCallback**) (int logLevel, const char *text, va_list args)
- typedef unsigned char *(* **LoadFileDataCallback**) (const char *fileName, int *dataSize)
- typedef **bool**(* **SaveFileDataCallback**) (const char *fileName, void *data, int dataSize)
- typedef char *(* **LoadFileTextCallback**) (const char *fileName)
- typedef **bool**(* **SaveFileTextCallback**) (const char *fileName, char *text)
- typedef void(* **AudioCallback**) (void *bufferData, unsigned int frames)

Enumerações

- enum **bool** { **false** = 0, **true** = !false }
- enum **ConfigFlags** { **FLAG_VSYNC_HINT** = 0x00000040, **FLAG_FULLSCREEN_MODE** = 0x00000002, **FLAG_WINDOW_RESIZABLE** = 0x00000004, **FLAG_WINDOW_UNDECORATED** = 0x00000008, **FLAG_WINDOW_HIDDEN** = 0x00000080, **FLAG_WINDOW_MINIMIZED** = 0x00000200, **FLAG_WINDOW_MAXIMIZED** = 0x00000400, **FLAG_WINDOW_UNFOCUSED** = 0x00000800, **FLAG_WINDOW_TOPMOST** = 0x00001000, **FLAG_WINDOW_ALWAYS_RUN** = 0x00000100, **FLAG_WINDOW_TRANSPARENT** = 0x00000010, **FLAG_WINDOW_HIGHDPI** = 0x00002000, **FLAG_WINDOW_MOUSE_PASSTHROUGH** = 0x00004000, **FLAG_BORDERLESS_WINDOWED_MODE** = 0x00008000, **FLAG_MSAA_4X_HINT** = 0x00000020, **FLAG_INTERLACED_HINT** = 0x00010000 }
- enum **TraceLogLevel** { **LOG_ALL** = 0, **LOG_TRACE**, **LOG_DEBUG**, **LOG_INFO**, **LOG_WARNING**, **LOG_ERROR**, **LOG_FATAL**, **LOG_NONE** }
- enum **KeyboardKey** { **KEY_NULL** = 0, **KEY_APOSTROPHE** = 39, **KEY_COMMA** = 44, **KEY_MINUS** = 45, **KEY_PERIOD** = 46, **KEY_SLASH** = 47, **KEY_ZERO** = 48, **KEY_ONE** = 49, **KEY_TWO** = 50, **KEY_THREE** = 51, **KEY_FOUR** = 52, **KEY_FIVE** = 53, **KEY_SIX** = 54, **KEY_SEVEN** = 55, **KEY_EIGHT** = 56, **KEY_NINE** = 57, **KEY_SEMICOLON** = 59, **KEY_EQUAL** = 61, **KEY_A** = 65, **KEY_B** = 66, **KEY_C** = 67, **KEY_D** = 68, **KEY_E** = 69, **KEY_F** = 70, **KEY_G** = 71, **KEY_H** = 72, **KEY_I** = 73, **KEY_J** = 74, **KEY_K** = 75, **KEY_L** = 76, **KEY_M** = 77, **KEY_N** = 78, **KEY_O** = 79, **KEY_P** = 80, **KEY_Q** = 81, **KEY_R** = 82, **KEY_S** = 83, **KEY_T** = 84, **KEY_U** = 85, **KEY_V** = 86, **KEY_W** = 87, **KEY_X** = 88, **KEY_Y** = 89, **KEY_Z** = 90, **KEY_LEFT_BRACKET** = 91, **KEY_BACKSLASH** = 92, **KEY_RIGHT_BRACKET** = 93, **KEY_GRAVE** = 96, **KEY_SPACE** = 32, **KEY_ESCAPE** = 256, **KEY_ENTER** = 257, **KEY_TAB** = 258, **KEY_BACKSPACE** = 259, **KEY_INSERT** = 260, **KEY_DELETE** = 261, **KEY_RIGHT** = 262, **KEY_LEFT** = 263, **KEY_DOWN** = 264, **KEY_UP** = 265, **KEY_PAGE_UP** = 266, **KEY_PAGE_DOWN** = 267, **KEY_HOME** = 268, **KEY_END** = 269, **KEY_CAPS_LOCK** = 280, **KEY_SCROLL_LOCK** = 281, **KEY_NUM_LOCK** = 282, **KEY_PRINT_SCREEN** = 283, **KEY_PAUSE** = 284, **KEY_F1** = 290, **KEY_F2** = 291, **KEY_F3** = 292, **KEY_F4** = 293, **KEY_F5** = 294, **KEY_F6** = 295,

KEY_F7 = 296, **KEY_F8** = 297, **KEY_F9** = 298, **KEY_F10** = 299, **KEY_F11** = 300, **KEY_F12** = 301, **KEY_LEFT_SHIFT** = 340, **KEY_LEFT_CONTROL** = 341, **KEY_LEFT_ALT** = 342, **KEY_LEFT_SUPER** = 343, **KEY_RIGHT_SHIFT** = 344, **KEY_RIGHT_CONTROL** = 345, **KEY_RIGHT_ALT** = 346, **KEY_RIGHT_SUPER** = 347, **KEY_KB_MENU** = 348, **KEY_KP_0** = 320, **KEY_KP_1** = 321, **KEY_KP_2** = 322, **KEY_KP_3** = 323, **KEY_KP_4** = 324, **KEY_KP_5** = 325, **KEY_KP_6** = 326, **KEY_KP_7** = 327, **KEY_KP_8** = 328, **KEY_KP_9** = 329, **KEY_KP_DECIMAL** = 330, **KEY_KP_DIVIDE** = 331, **KEY_KP_MULTIPLY** = 332, **KEY_KP_SUBTRACT** = 333, **KEY_KP_ADD** = 334, **KEY_KP_ENTER** = 335, **KEY_KP_EQUAL** = 336, **KEY_BACK** = 4, **KEY_MENU** = 5, **KEY_VOLUME_UP** = 24, **KEY_VOLUME_DOWN** = 25 }

- enum **MouseButton** { **MOUSE_BUTTON_LEFT** = 0, **MOUSE_BUTTON_RIGHT** = 1, **MOUSE_BUTTON_MIDDLE** = 2, **MOUSE_BUTTON_SIDE** = 3, **MOUSE_BUTTON_EXTRA** = 4, **MOUSE_BUTTON_FORWARD** = 5, **MOUSE_BUTTON_BACK** = 6 }
- enum **MouseCursor** { **MOUSE_CURSOR_DEFAULT** = 0, **MOUSE_CURSOR_ARROW** = 1, **MOUSE_CURSOR_IBEAM** = 2, **MOUSE_CURSOR_CROSSHAIR** = 3, **MOUSE_CURSOR_POINTING_HAND** = 4, **MOUSE_CURSOR_RESIZE_EW** = 5, **MOUSE_CURSOR_RESIZE_NS** = 6, **MOUSE_CURSOR_RESIZE_NWSE** = 7, **MOUSE_CURSOR_RESIZE_NESW** = 8, **MOUSE_CURSOR_RESIZE_ALL** = 9, **MOUSE_CURSOR_NOT_ALLOWED** = 10 }
- enum **GamepadButton** { **GAMEPAD_BUTTON_UNKNOWN** = 0, **GAMEPAD_BUTTON_LEFT_FACE_UP**, **GAMEPAD_BUTTON_LEFT_FACE_RIGHT**, **GAMEPAD_BUTTON_LEFT_FACE_DOWN**, **GAMEPAD_BUTTON_LEFT_FACE_LEFT**, **GAMEPAD_BUTTON_RIGHT_FACE_UP**, **GAMEPAD_BUTTON_RIGHT_FACE_RIGHT**, **GAMEPAD_BUTTON_RIGHT_FACE_DOWN**, **GAMEPAD_BUTTON_RIGHT_FACE_LEFT**, **GAMEPAD_BUTTON_LEFT_TRIGGER_1**, **GAMEPAD_BUTTON_LEFT_TRIGGER_2**, **GAMEPAD_BUTTON_RIGHT_TRIGGER_1**, **GAMEPAD_BUTTON_RIGHT_TRIGGER_2**, **GAMEPAD_BUTTON_MIDDLE_LEFT**, **GAMEPAD_BUTTON_MIDDLE**, **GAMEPAD_BUTTON_MIDDLE_RIGHT**, **GAMEPAD_BUTTON_LEFT_THUMB**, **GAMEPAD_BUTTON_RIGHT_THUMB** }
- enum **GamepadAxis** { **GAMEPAD_AXIS_LEFT_X** = 0, **GAMEPAD_AXIS_LEFT_Y** = 1, **GAMEPAD_AXIS_RIGHT_X** = 2, **GAMEPAD_AXIS_RIGHT_Y** = 3, **GAMEPAD_AXIS_LEFT_TRIGGER** = 4, **GAMEPAD_AXIS_RIGHT_TRIGGER** = 5 }
- enum **MaterialMapIndex** { **MATERIAL_MAP_ALBEDO** = 0, **MATERIAL_MAP_METALNESS**, **MATERIAL_MAP_NORMAL**, **MATERIAL_MAP_ROUGHNESS**, **MATERIAL_MAP_OCCLUSION**, **MATERIAL_MAP_EMISSION**, **MATERIAL_MAP_HEIGHT**, **MATERIAL_MAP_CUBEMAP**, **MATERIAL_MAP_IRRADIANCE**, **MATERIAL_MAP_PREFILTER**, **MATERIAL_MAP_BRDF** }
- enum **ShaderLocationIndex** { **SHADER_LOC_VERTEX_POSITION** = 0, **SHADER_LOC_VERTEX_TEXCOORD01**, **SHADER_LOC_VERTEX_TEXCOORD02**, **SHADER_LOC_VERTEX_NORMAL**, **SHADER_LOC_VERTEX_TANGENT**, **SHADER_LOC_VERTEX_COLOR**, **SHADER_LOC_MATRIX_MVP**, **SHADER_LOC_MATRIX_VIEW**, **SHADER_LOC_MATRIX_PROJECTION**, **SHADER_LOC_MATRIX_MODEL**, **SHADER_LOC_MATRIX_NORMAL**, **SHADER_LOC_VECTOR_VIEW**, **SHADER_LOC_COLOR_DIFFUSE**, **SHADER_LOC_COLOR_SPECULAR**, **SHADER_LOC_COLOR_AMBIENT**, **SHADER_LOC_MAP_ALBEDO**, **SHADER_LOC_MAP_METALNESS**, **SHADER_LOC_MAP_NORMAL**, **SHADER_LOC_MAP_ROUGHNESS**, **SHADER_LOC_MAP_OCCLUSION**, **SHADER_LOC_MAP_EMISSION**, **SHADER_LOC_MAP_HEIGHT**, **SHADER_LOC_MAP_CUBEMAP**, **SHADER_LOC_MAP_IRRADIANCE**, **SHADER_LOC_MAP_PREFILTER**, **SHADER_LOC_MAP_BRDF**, **SHADER_LOC_VERTEX_BONEIDS**, **SHADER_LOC_VERTEX_BONEWEIGHTS**, **SHADER_LOC_BONE_MATRICES** }
- enum **ShaderUniformDataType** { **SHADER_UNIFORM_FLOAT** = 0, **SHADER_UNIFORM_VEC2**, **SHADER_UNIFORM_VEC3**, **SHADER_UNIFORM_VEC4**, **SHADER_UNIFORM_INT**, **SHADER_UNIFORM_IVEC2**, **SHADER_UNIFORM_IVEC3**, **SHADER_UNIFORM_IVEC4**, **SHADER_UNIFORM_SAMPLER2D** }

- enum **ShaderAttributeDataType** { **SHADER_ATTRIB_FLOAT** = 0, **SHADER_ATTRIB_VEC2**, **SHADER_ATTRIB_VEC3**, **SHADER_ATTRIB_VEC4** }
- enum **PixelFormat** { **PIXELFORMAT_UNCOMPRESSED_GRAYSCALE** = 1, **PIXELFORMAT_UNCOMPRESSED_GRAY_ALPHA**, **PIXELFORMAT_UNCOMPRESSED_R5G6B5**, **PIXELFORMAT_UNCOMPRESSED_R8G8B8**, **PIXELFORMAT_UNCOMPRESSED_R5G5B5A1**, **PIXELFORMAT_UNCOMPRESSED_R4G4B4A4**, **PIXELFORMAT_UNCOMPRESSED_R8G8B8A8**, **PIXELFORMAT_UNCOMPRESSED_R32**, **PIXELFORMAT_UNCOMPRESSED_R32G32B32**, **PIXELFORMAT_UNCOMPRESSED_R32G32B32A32**, **PIXELFORMAT_UNCOMPRESSED_R16**, **PIXELFORMAT_UNCOMPRESSED_R16G16B16**, **PIXELFORMAT_UNCOMPRESSED_R16G16B16A16**, **PIXELFORMAT_COMPRESSED_DXT1_RGB**, **PIXELFORMAT_COMPRESSED_DXT1_RGBA**, **PIXELFORMAT_COMPRESSED_DXT3_RGBA**, **PIXELFORMAT_COMPRESSED_DXT5_RGBA**, **PIXELFORMAT_COMPRESSED_ETC1_RGB**, **PIXELFORMAT_COMPRESSED_ETC2_RGB**, **PIXELFORMAT_COMPRESSED_ETC2_EAC_RGBA**, **PIXELFORMAT_COMPRESSED_PVRT_RGB**, **PIXELFORMAT_COMPRESSED_PVRT_RGBA**, **PIXELFORMAT_COMPRESSED_ASTC_4x4_RGBA**, **PIXELFORMAT_COMPRESSED_ASTC_8x8_RGBA** }
- enum **TextureFilter** { **TEXTURE_FILTER_POINT** = 0, **TEXTURE_FILTER_BILINEAR**, **TEXTURE_FILTER_TRILINEAR**, **TEXTURE_FILTER_ANISOTROPIC_4X**, **TEXTURE_FILTER_ANISOTROPIC_8X**, **TEXTURE_FILTER_ANISOTROPIC_16X** }
- enum **TextureWrap** { **TEXTURE_WRAP_REPEAT** = 0, **TEXTURE_WRAP_CLAMP**, **TEXTURE_WRAP_MIRROR_REPEAT**, **TEXTURE_WRAP_MIRROR_CLAMP** }
- enum **CubemapLayout** { **CUBEMAP_LAYOUT_AUTO_DETECT** = 0, **CUBEMAP_LAYOUT_LINE_VERTICAL**, **CUBEMAP_LAYOUT_LINE_HORIZONTAL**, **CUBEMAP_LAYOUT_CROSS_THREE_BY_FOUR**, **CUBEMAP_LAYOUT_CROSS_FOUR_BY_THREE** }
- enum **FontType** { **FONT_DEFAULT** = 0, **FONT_BITMAP**, **FONT_SDF** }
- enum **BlendMode** { **BLEND_ALPHA** = 0, **BLEND_ADDITIVE**, **BLEND_MULTIPLIED**, **BLEND_ADD_COLORS**, **BLEND_SUBTRACT_COLORS**, **BLEND_ALPHA_PREMULTIPLY**, **BLEND_CUSTOM**, **BLEND_CUSTOM_SEPARATE** }
- enum **Gesture** { **GESTURE_NONE** = 0, **GESTURE_TAP** = 1, **GESTURE_DOUBLETAP** = 2, **GESTURE_HOLD** = 4, **GESTURE_DRAG** = 8, **GESTURE_SWIPE_RIGHT** = 16, **GESTURE_SWIPE_LEFT** = 32, **GESTURE_SWIPE_UP** = 64, **GESTURE_SWIPE_DOWN** = 128, **GESTURE_PINCH_IN** = 256, **GESTURE_PINCH_OUT** = 512 }
- enum **CameraMode** { **CAMERA_CUSTOM** = 0, **CAMERA_FREE**, **CAMERA_ORBITAL**, **CAMERA_FIRST_PERSON**, **CAMERA_THIRD_PERSON** }
- enum **CameraProjection** { **CAMERA_PERSPECTIVE** = 0, **CAMERA_ORTHOGRAPHIC** }
- enum **NPatchLayout** { **NPATCH_NINE_PATCH** = 0, **NPATCH_THREE_PATCH_VERTICAL**, **NPATCH_THREE_PATCH_HORIZONTAL** }

Funções

- **RLAPI** void **InitWindow** (int width, int height, const char *title)
- **RLAPI** void **CloseWindow** (void)
- **RLAPI bool WindowShouldClose** (void)
- **RLAPI bool IsWindowReady** (void)
- **RLAPI bool IsWindowFullscreen** (void)
- **RLAPI bool IsWindowHidden** (void)
- **RLAPI bool IsWindowMinimized** (void)
- **RLAPI bool IsWindowMaximized** (void)
- **RLAPI bool IsWindowFocused** (void)
- **RLAPI bool IsWindowResized** (void)

- **RLAPI bool IsWindowState** (unsigned int flag)
- **RLAPI** void **SetWindowState** (unsigned int flags)
- **RLAPI** void **ClearWindowState** (unsigned int flags)
- **RLAPI** void **ToggleFullscreen** (void)
- **RLAPI** void **ToggleBorderlessWindowed** (void)
- **RLAPI** void **MaximizeWindow** (void)
- **RLAPI** void **MinimizeWindow** (void)
- **RLAPI** void **RestoreWindow** (void)
- **RLAPI** void **SetWindowIcon** (**Image** image)
- **RLAPI** void **SetWindowIcons** (**Image** *images, int count)
- **RLAPI** void **SetWindowTitle** (const char *title)
- **RLAPI** void **SetWindowPosition** (int x, int y)
- **RLAPI** void **SetWindowMonitor** (int monitor)
- **RLAPI** void **SetWindowMinSize** (int width, int height)
- **RLAPI** void **SetWindowMaxSize** (int width, int height)
- **RLAPI** void **SetWindowSize** (int width, int height)
- **RLAPI** void **SetWindowOpacity** (float opacity)
- **RLAPI** void **SetWindowFocused** (void)
- **RLAPI** void * **GetWindowHandle** (void)
- **RLAPI** int **GetScreenWidth** (void)
- **RLAPI** int **GetScreenHeight** (void)
- **RLAPI** int **GetRenderWidth** (void)
- **RLAPI** int **GetRenderHeight** (void)
- **RLAPI** int **GetMonitorCount** (void)
- **RLAPI** int **GetCurrentMonitor** (void)
- **RLAPI Vector2 GetMonitorPosition** (int monitor)
- **RLAPI** int **GetMonitorWidth** (int monitor)
- **RLAPI** int **GetMonitorHeight** (int monitor)
- **RLAPI** int **GetMonitorPhysicalWidth** (int monitor)
- **RLAPI** int **GetMonitorPhysicalHeight** (int monitor)
- **RLAPI** int **GetMonitorRefreshRate** (int monitor)
- **RLAPI Vector2 GetWindowPosition** (void)
- **RLAPI Vector2 GetWindowScaleDPI** (void)
- **RLAPI** const char * **GetMonitorName** (int monitor)
- **RLAPI** void **SetClipboardText** (const char *text)
- **RLAPI** const char * **GetClipboardText** (void)
- **RLAPI Image GetClipboardImage** (void)
- **RLAPI** void **EnableEventWaiting** (void)
- **RLAPI** void **DisableEventWaiting** (void)
- **RLAPI** void **ShowCursor** (void)
- **RLAPI** void **HideCursor** (void)
- **RLAPI bool IsCursorHidden** (void)
- **RLAPI** void **EnableCursor** (void)
- **RLAPI** void **DisableCursor** (void)
- **RLAPI bool IsCursorOnScreen** (void)
- **RLAPI** void **ClearBackground** (**Color** color)
- **RLAPI** void **BeginDrawing** (void)
- **RLAPI** void **EndDrawing** (void)
- **RLAPI** void **BeginMode2D** (**Camera2D** camera)
- **RLAPI** void **EndMode2D** (void)
- **RLAPI** void **BeginMode3D** (**Camera3D** camera)
- **RLAPI** void **EndMode3D** (void)
- **RLAPI** void **BeginTextureMode** (**RenderTexture2D** target)
- **RLAPI** void **EndTextureMode** (void)
- **RLAPI** void **BeginShaderMode** (**Shader** shader)
- **RLAPI** void **EndShaderMode** (void)
- **RLAPI** void **BeginBlendMode** (int mode)
- **RLAPI** void **EndBlendMode** (void)
- **RLAPI** void **BeginScissorMode** (int x, int y, int width, int height)
- **RLAPI** void **EndScissorMode** (void)

- **RLAPI** void **BeginVrStereoMode** (**VrStereoConfig** config)
- **RLAPI** void **EndVrStereoMode** (void)
- **RLAPI VrStereoConfig LoadVrStereoConfig** (**VrDeviceInfo** device)
- **RLAPI** void **UnloadVrStereoConfig** (**VrStereoConfig** config)
- **RLAPI Shader LoadShader** (const char *vsFileName, const char *fsFileName)
- **RLAPI Shader LoadShaderFromMemory** (const char *vsCode, const char *fsCode)
- **RLAPI bool IsShaderValid** (**Shader** shader)
- **RLAPI** int **GetShaderLocation** (**Shader** shader, const char *uniformName)
- **RLAPI** int **GetShaderLocationAttrib** (**Shader** shader, const char *attribName)
- **RLAPI** void **SetShaderValue** (**Shader** shader, int locIndex, const void *value, int uniformType)
- **RLAPI** void **SetShaderValueV** (**Shader** shader, int locIndex, const void *value, int uniformType, int count)
- **RLAPI** void **SetShaderValueMatrix** (**Shader** shader, int locIndex, **Matrix** mat)
- **RLAPI** void **SetShaderValueTexture** (**Shader** shader, int locIndex, **Texture2D** texture)
- **RLAPI** void **UnloadShader** (**Shader** shader)
- **RLAPI Ray GetScreenToWorldRay** (**Vector2** position, **Camera** camera)
- **RLAPI Ray GetScreenToWorldRayEx** (**Vector2** position, **Camera** camera, int width, int height)
- **RLAPI Vector2 GetWorldToScreen** (**Vector3** position, **Camera** camera)
- **RLAPI Vector2 GetWorldToScreenEx** (**Vector3** position, **Camera** camera, int width, int height)
- **RLAPI Vector2 GetWorldToScreen2D** (**Vector2** position, **Camera2D** camera)
- **RLAPI Vector2 GetScreenToWorld2D** (**Vector2** position, **Camera2D** camera)
- **RLAPI Matrix GetCameraMatrix** (**Camera** camera)
- **RLAPI Matrix GetCameraMatrix2D** (**Camera2D** camera)
- **RLAPI** void **SetTargetFPS** (int fps)
- **RLAPI** float **GetFrameTime** (void)
- **RLAPI** double **GetTime** (void)
- **RLAPI** int **GetFPS** (void)
- **RLAPI** void **SwapScreenBuffer** (void)
- **RLAPI** void **PollInputEvents** (void)
- **RLAPI** void **WaitTime** (double seconds)
- **RLAPI** void **SetRandomSeed** (unsigned int seed)
- **RLAPI** int **GetRandomValue** (int min, int max)
- **RLAPI** int * **LoadRandomSequence** (unsigned int count, int min, int max)
- **RLAPI** void **UnloadRandomSequence** (int *sequence)
- **RLAPI** void **TakeScreenshot** (const char *fileName)
- **RLAPI** void **SetConfigFlags** (unsigned int flags)
- **RLAPI** void **OpenURL** (const char *url)
- **RLAPI** void **TraceLog** (int logLevel, const char *text,...)
- **RLAPI** void **SetTraceLogLevel** (int logLevel)
- **RLAPI** void * **MemAlloc** (unsigned int size)
- **RLAPI** void * **MemRealloc** (void *ptr, unsigned int size)
- **RLAPI** void **MemFree** (void *ptr)
- **RLAPI** void **SetTraceLogCallback** (**TraceLogCallback** callback)
- **RLAPI** void **SetLoadFileDataCallback** (**LoadFileDataCallback** callback)
- **RLAPI** void **SetSaveFileDataCallback** (**SaveFileDataCallback** callback)
- **RLAPI** void **SetLoadFileTextCallback** (**LoadFileTextCallback** callback)
- **RLAPI** void **SetSaveFileTextCallback** (**SaveFileTextCallback** callback)
- **RLAPI** unsigned char * **LoadFileData** (const char *fileName, int *dataSize)
- **RLAPI** void **UnloadFileData** (unsigned char *data)
- **RLAPI bool SaveFileData** (const char *fileName, void *data, int dataSize)
- **RLAPI bool ExportDataAsCode** (const unsigned char *data, int dataSize, const char *fileName)
- **RLAPI** char * **LoadFileText** (const char *fileName)
- **RLAPI** void **UnloadFileText** (char *text)
- **RLAPI bool SaveFileText** (const char *fileName, char *text)
- **RLAPI bool FileExists** (const char *fileName)
- **RLAPI bool DirectoryExists** (const char *dirPath)
- **RLAPI bool IsFileExtension** (const char *fileName, const char *ext)
- **RLAPI** int **GetFileLength** (const char *fileName)

- **RLAPI** const char * **GetFileExtension** (const char *fileName)
- **RLAPI** const char * **GetFileName** (const char *filePath)
- **RLAPI** const char * **GetFileNameWithoutExt** (const char *filePath)
- **RLAPI** const char * **GetDirectoryPath** (const char *filePath)
- **RLAPI** const char * **GetPrevDirectoryPath** (const char *dirPath)
- **RLAPI** const char * **GetWorkingDirectory** (void)
- **RLAPI** const char * **GetApplicationDirectory** (void)
- **RLAPI** int **MakeDirectory** (const char *dirPath)
- **RLAPI bool ChangeDirectory** (const char *dir)
- **RLAPI bool IsPathFile** (const char *path)
- **RLAPI bool IsFileNameValid** (const char *fileName)
- **RLAPI FilePathList LoadDirectoryFiles** (const char *dirPath)
- **RLAPI FilePathList LoadDirectoryFilesEx** (const char *basePath, const char *filter, **bool** scanSubdirs)
- **RLAPI** void **UnloadDirectoryFiles** (**FilePathList** files)
- **RLAPI bool IsFileDropped** (void)
- **RLAPI FilePathList LoadDroppedFiles** (void)
- **RLAPI** void **UnloadDroppedFiles** (**FilePathList** files)
- **RLAPI** long **GetFileModTime** (const char *fileName)
- **RLAPI** unsigned char * **CompressData** (const unsigned char *data, int dataSize, int *compDataSize)
- **RLAPI** unsigned char * **DecompressData** (const unsigned char *compData, int compDataSize, int *dataSize)
- **RLAPI** char * **EncodeDataBase64** (const unsigned char *data, int dataSize, int *outputSize)
- **RLAPI** unsigned char * **DecodeDataBase64** (const unsigned char *data, int *outputSize)
- **RLAPI** unsigned int **ComputeCRC32** (unsigned char *data, int dataSize)
- **RLAPI** unsigned int * **ComputeMD5** (unsigned char *data, int dataSize)
- **RLAPI** unsigned int * **ComputeSHA1** (unsigned char *data, int dataSize)
- **RLAPI AutomationEventList LoadAutomationEventList** (const char *fileName)
- **RLAPI** void **UnloadAutomationEventList** (**AutomationEventList** list)
- **RLAPI bool ExportAutomationEventList** (**AutomationEventList** list, const char *fileName)
- **RLAPI** void **SetAutomationEventList** (**AutomationEventList** *list)
- **RLAPI** void **SetAutomationEventBaseFrame** (int frame)
- **RLAPI** void **StartAutomationEventRecording** (void)
- **RLAPI** void **StopAutomationEventRecording** (void)
- **RLAPI** void **PlayAutomationEvent** (**AutomationEvent** event)
- **RLAPI bool IsKeyPressed** (int key)
- **RLAPI bool IsKeyPressedRepeat** (int key)
- **RLAPI bool IsKeyDown** (int key)
- **RLAPI bool IsKeyReleased** (int key)
- **RLAPI bool IsKeyUp** (int key)
- **RLAPI** int **GetKeyPressed** (void)
- **RLAPI** int **GetCharPressed** (void)
- **RLAPI** void **SetExitKey** (int key)
- **RLAPI bool IsGamepadAvailable** (int gamepad)
- **RLAPI** const char * **GetGamepadName** (int gamepad)
- **RLAPI bool IsGamepadButtonPressed** (int gamepad, int button)
- **RLAPI bool IsGamepadButtonDown** (int gamepad, int button)
- **RLAPI bool IsGamepadButtonReleased** (int gamepad, int button)
- **RLAPI bool IsGamepadButtonUp** (int gamepad, int button)
- **RLAPI** int **GetGamepadButtonPressed** (void)
- **RLAPI** int **GetGamepadAxisCount** (int gamepad)
- **RLAPI** float **GetGamepadAxisMovement** (int gamepad, int axis)
- **RLAPI** int **SetGamepadMappings** (const char *mappings)
- **RLAPI** void **SetGamepadVibration** (int gamepad, float leftMotor, float rightMotor, float duration)
- **RLAPI bool IsMouseButtonPressed** (int button)
- **RLAPI bool IsMouseButtonDown** (int button)
- **RLAPI bool IsMouseButtonReleased** (int button)
- **RLAPI bool IsMouseButtonUp** (int button)

- **RLAPI** int **GetMouseX** (void)
- **RLAPI** int **GetMouseY** (void)
- **RLAPI Vector2 GetMousePosition** (void)
- **RLAPI Vector2 GetMouseDelta** (void)
- **RLAPI** void **SetMousePosition** (int x, int y)
- **RLAPI** void **SetMouseOffset** (int offsetX, int offsetY)
- **RLAPI** void **SetMouseScale** (float scaleX, float scaleY)
- **RLAPI** float **GetMouseWheelMove** (void)
- **RLAPI Vector2 GetMouseWheelMoveV** (void)
- **RLAPI** void **SetMouseCursor** (int cursor)
- **RLAPI** int **GetTouchX** (void)
- **RLAPI** int **GetTouchY** (void)
- **RLAPI Vector2 GetTouchPosition** (int index)
- **RLAPI** int **GetTouchPointId** (int index)
- **RLAPI** int **GetTouchPointCount** (void)
- **RLAPI** void **SetGesturesEnabled** (unsigned int flags)
- **RLAPI bool IsGestureDetected** (unsigned int gesture)
- **RLAPI** int **GetGestureDetected** (void)
- **RLAPI** float **GetGestureHoldDuration** (void)
- **RLAPI Vector2 GetGestureDragVector** (void)
- **RLAPI** float **GetGestureDragAngle** (void)
- **RLAPI Vector2 GetGesturePinchVector** (void)
- **RLAPI** float **GetGesturePinchAngle** (void)
- **RLAPI** void **UpdateCamera** (**Camera** *camera, int mode)
- **RLAPI** void **UpdateCameraPro** (**Camera** *camera, **Vector3** movement, **Vector3** rotation, float zoom)
- **RLAPI** void **SetShapesTexture** (**Texture2D** texture, **Rectangle** source)
- **RLAPI Texture2D GetShapesTexture** (void)
- **RLAPI Rectangle GetShapesTextureRectangle** (void)
- **RLAPI** void **DrawPixel** (int posX, int posY, **Color** color)
- **RLAPI** void **DrawPixelV** (**Vector2** position, **Color** color)
- **RLAPI** void **DrawLine** (int startPosX, int startPosY, int endPosX, int endPosY, **Color** color)
- **RLAPI** void **DrawLineV** (**Vector2** startPos, **Vector2** endPos, **Color** color)
- **RLAPI** void **DrawLineEx** (**Vector2** startPos, **Vector2** endPos, float thick, **Color** color)
- **RLAPI** void **DrawLineStrip** (const **Vector2** *points, int pointCount, **Color** color)
- **RLAPI** void **DrawLineBezier** (**Vector2** startPos, **Vector2** endPos, float thick, **Color** color)
- **RLAPI** void **DrawCircle** (int centerX, int centerY, float radius, **Color** color)
- **RLAPI** void **DrawCircleSector** (**Vector2** center, float radius, float startAngle, float endAngle, int segments, **Color** color)
- **RLAPI** void **DrawCircleSectorLines** (**Vector2** center, float radius, float startAngle, float endAngle, int segments, **Color** color)
- **RLAPI** void **DrawCircleGradient** (int centerX, int centerY, float radius, **Color** inner, **Color** outer)
- **RLAPI** void **DrawCircleV** (**Vector2** center, float radius, **Color** color)
- **RLAPI** void **DrawCircleLines** (int centerX, int centerY, float radius, **Color** color)
- **RLAPI** void **DrawCircleLinesV** (**Vector2** center, float radius, **Color** color)
- **RLAPI** void **DrawEllipse** (int centerX, int centerY, float radiusH, float radiusV, **Color** color)
- **RLAPI** void **DrawEllipseLines** (int centerX, int centerY, float radiusH, float radiusV, **Color** color)
- **RLAPI** void **DrawRing** (**Vector2** center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, **Color** color)
- **RLAPI** void **DrawRingLines** (**Vector2** center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, **Color** color)
- **RLAPI** void **DrawRectangle** (int posX, int posY, int width, int height, **Color** color)
- **RLAPI** void **DrawRectangleV** (**Vector2** position, **Vector2** size, **Color** color)
- **RLAPI** void **DrawRectangleRec** (**Rectangle** rec, **Color** color)
- **RLAPI** void **DrawRectanglePro** (**Rectangle** rec, **Vector2** origin, float rotation, **Color** color)
- **RLAPI** void **DrawRectangleGradientV** (int posX, int posY, int width, int height, **Color** top, **Color** bottom)

- **RLAPI** void **DrawRectangleGradientH** (int posX, int posY, int width, int height, **Color** left, **Color** right)
- **RLAPI** void **DrawRectangleGradientEx** (**Rectangle** rec, **Color** topLeft, **Color** bottomLeft, **Color** topRight, **Color** bottomRight)
- **RLAPI** void **DrawRectangleLines** (int posX, int posY, int width, int height, **Color** color)
- **RLAPI** void **DrawRectangleLinesEx** (**Rectangle** rec, float lineThick, **Color** color)
- **RLAPI** void **DrawRectangleRounded** (**Rectangle** rec, float roundness, int segments, **Color** color)
- **RLAPI** void **DrawRectangleRoundedLines** (**Rectangle** rec, float roundness, int segments, **Color** color)
- **RLAPI** void **DrawRectangleRoundedLinesEx** (**Rectangle** rec, float roundness, int segments, float lineThick, **Color** color)
- **RLAPI** void **DrawTriangle** (**Vector2** v1, **Vector2** v2, **Vector2** v3, **Color** color)
- **RLAPI** void **DrawTriangleLines** (**Vector2** v1, **Vector2** v2, **Vector2** v3, **Color** color)
- **RLAPI** void **DrawTriangleFan** (const **Vector2** *points, int pointCount, **Color** color)
- **RLAPI** void **DrawTriangleStrip** (const **Vector2** *points, int pointCount, **Color** color)
- **RLAPI** void **DrawPoly** (**Vector2** center, int sides, float radius, float rotation, **Color** color)
- **RLAPI** void **DrawPolyLines** (**Vector2** center, int sides, float radius, float rotation, **Color** color)
- **RLAPI** void **DrawPolyLinesEx** (**Vector2** center, int sides, float radius, float rotation, float lineThick, **Color** color)
- **RLAPI** void **DrawSplineLinear** (const **Vector2** *points, int pointCount, float thick, **Color** color)
- **RLAPI** void **DrawSplineBasis** (const **Vector2** *points, int pointCount, float thick, **Color** color)
- **RLAPI** void **DrawSplineCatmullRom** (const **Vector2** *points, int pointCount, float thick, **Color** color)
- **RLAPI** void **DrawSplineBezierQuadratic** (const **Vector2** *points, int pointCount, float thick, **Color** color)
- **RLAPI** void **DrawSplineBezierCubic** (const **Vector2** *points, int pointCount, float thick, **Color** color)
- **RLAPI** void **DrawSplineSegmentLinear** (**Vector2** p1, **Vector2** p2, float thick, **Color** color)
- **RLAPI** void **DrawSplineSegmentBasis** (**Vector2** p1, **Vector2** p2, **Vector2** p3, **Vector2** p4, float thick, **Color** color)
- **RLAPI** void **DrawSplineSegmentCatmullRom** (**Vector2** p1, **Vector2** p2, **Vector2** p3, **Vector2** p4, float thick, **Color** color)
- **RLAPI** void **DrawSplineSegmentBezierQuadratic** (**Vector2** p1, **Vector2** c2, **Vector2** p3, float thick, **Color** color)
- **RLAPI** void **DrawSplineSegmentBezierCubic** (**Vector2** p1, **Vector2** c2, **Vector2** c3, **Vector2** p4, float thick, **Color** color)
- **RLAPI Vector2 GetSplinePointLinear** (**Vector2** startPos, **Vector2** endPos, float t)
- **RLAPI Vector2 GetSplinePointBasis** (**Vector2** p1, **Vector2** p2, **Vector2** p3, **Vector2** p4, float t)
- **RLAPI Vector2 GetSplinePointCatmullRom** (**Vector2** p1, **Vector2** p2, **Vector2** p3, **Vector2** p4, float t)
- **RLAPI Vector2 GetSplinePointBezierQuad** (**Vector2** p1, **Vector2** c2, **Vector2** p3, float t)
- **RLAPI Vector2 GetSplinePointBezierCubic** (**Vector2** p1, **Vector2** c2, **Vector2** c3, **Vector2** p4, float t)
- **RLAPI bool CheckCollisionRecs** (**Rectangle** rec1, **Rectangle** rec2)
- **RLAPI bool CheckCollisionCircles** (**Vector2** center1, float radius1, **Vector2** center2, float radius2)
- **RLAPI bool CheckCollisionCircleRec** (**Vector2** center, float radius, **Rectangle** rec)
- **RLAPI bool CheckCollisionCircleLine** (**Vector2** center, float radius, **Vector2** p1, **Vector2** p2)
- **RLAPI bool CheckCollisionPointRec** (**Vector2** point, **Rectangle** rec)
- **RLAPI bool CheckCollisionPointCircle** (**Vector2** point, **Vector2** center, float radius)
- **RLAPI bool CheckCollisionPointTriangle** (**Vector2** point, **Vector2** p1, **Vector2** p2, **Vector2** p3)
- **RLAPI bool CheckCollisionPointLine** (**Vector2** point, **Vector2** p1, **Vector2** p2, int threshold)
- **RLAPI bool CheckCollisionPointPoly** (**Vector2** point, const **Vector2** *points, int pointCount)
- **RLAPI bool CheckCollisionLines** (**Vector2** startPos1, **Vector2** endPos1, **Vector2** startPos2, **Vector2** endPos2, **Vector2** *collisionPoint)
- **RLAPI Rectangle GetCollisionRec** (**Rectangle** rec1, **Rectangle** rec2)
- **RLAPI Image LoadImage** (const char *fileName)

- **RLAPI Image LoadImageRaw** (const char *fileName, int width, int height, int format, int headerSize)
- **RLAPI Image LoadImageAnim** (const char *fileName, int *frames)
- **RLAPI Image LoadImageAnimFromMemory** (const char *fileType, const unsigned char *fileData, int dataSize, int *frames)
- **RLAPI Image LoadImageFromMemory** (const char *fileType, const unsigned char *fileData, int dataSize)
- **RLAPI Image LoadImageFromTexture** (**Texture2D** texture)
- **RLAPI Image LoadImageFromScreen** (void)
- **RLAPI bool IsImageValid** (**Image** image)
- **RLAPI** void **UnloadImage** (**Image** image)
- **RLAPI bool ExportImage** (**Image** image, const char *fileName)
- **RLAPI** unsigned char * **ExportImageToMemory** (**Image** image, const char *fileType, int *fileSize)
- **RLAPI bool ExportImageAsCode** (**Image** image, const char *fileName)
- **RLAPI Image GenImageColor** (int width, int height, **Color** color)
- **RLAPI Image GenImageGradientLinear** (int width, int height, int direction, **Color** start, **Color** end)
- **RLAPI Image GenImageGradientRadial** (int width, int height, float density, **Color** inner, **Color** outer)
- **RLAPI Image GenImageGradientSquare** (int width, int height, float density, **Color** inner, **Color** outer)
- **RLAPI Image GenImageChecked** (int width, int height, int checksX, int checksY, **Color** col1, **Color** col2)
- **RLAPI Image GenImageWhiteNoise** (int width, int height, float factor)
- **RLAPI Image GenImagePerlinNoise** (int width, int height, int offsetX, int offsetY, float scale)
- **RLAPI Image GenImageCellular** (int width, int height, int tileSize)
- **RLAPI Image GenImageText** (int width, int height, const char *text)
- **RLAPI Image ImageCopy** (**Image** image)
- **RLAPI Image ImageFromImage** (**Image** image, **Rectangle** rec)
- **RLAPI Image ImageFromChannel** (**Image** image, int selectedChannel)
- **RLAPI Image ImageText** (const char *text, int fontSize, **Color** color)
- **RLAPI Image ImageTextEx** (**Font** font, const char *text, float fontSize, float spacing, **Color** tint)
- **RLAPI** void **ImageFormat** (**Image** *image, int newFormat)
- **RLAPI** void **ImageToPOT** (**Image** *image, **Color** fill)
- **RLAPI** void **ImageCrop** (**Image** *image, **Rectangle** crop)
- **RLAPI** void **ImageAlphaCrop** (**Image** *image, float threshold)
- **RLAPI** void **ImageAlphaClear** (**Image** *image, **Color** color, float threshold)
- **RLAPI** void **ImageAlphaMask** (**Image** *image, **Image** alphaMask)
- **RLAPI** void **ImageAlphaPremultiply** (**Image** *image)
- **RLAPI** void **ImageBlurGaussian** (**Image** *image, int blurSize)
- **RLAPI** void **ImageKernelConvolution** (**Image** *image, const float *kernel, int kernelSize)
- **RLAPI** void **ImageResize** (**Image** *image, int newWidth, int newHeight)
- **RLAPI** void **ImageResizeNN** (**Image** *image, int newWidth, int newHeight)
- **RLAPI** void **ImageResizeCanvas** (**Image** *image, int newWidth, int newHeight, int offsetX, int offsetY, **Color** fill)
- **RLAPI** void **ImageMipmaps** (**Image** *image)
- **RLAPI** void **ImageDither** (**Image** *image, int rBpp, int gBpp, int bBpp, int aBpp)
- **RLAPI** void **ImageFlipVertical** (**Image** *image)
- **RLAPI** void **ImageFlipHorizontal** (**Image** *image)
- **RLAPI** void **ImageRotate** (**Image** *image, int degrees)
- **RLAPI** void **ImageRotateCW** (**Image** *image)
- **RLAPI** void **ImageRotateCCW** (**Image** *image)
- **RLAPI** void **ImageColorTint** (**Image** *image, **Color** color)
- **RLAPI** void **ImageColorInvert** (**Image** *image)
- **RLAPI** void **ImageColorGrayscale** (**Image** *image)
- **RLAPI** void **ImageColorContrast** (**Image** *image, float contrast)
- **RLAPI** void **ImageColorBrightness** (**Image** *image, int brightness)
- **RLAPI** void **ImageColorReplace** (**Image** *image, **Color** color, **Color** replace)

- **RLAPI Color** * **LoadImageColors** (**Image** image)
- **RLAPI Color** * **LoadImagePalette** (**Image** image, int maxPaletteSize, int *colorCount)
- **RLAPI** void **UnloadImageColors** (**Color** *colors)
- **RLAPI** void **UnloadImagePalette** (**Color** *colors)
- **RLAPI Rectangle GetImageAlphaBorder** (**Image** image, float threshold)
- **RLAPI Color GetImageColor** (**Image** image, int x, int y)
- **RLAPI** void **ImageClearBackground** (**Image** *dst, **Color** color)
- **RLAPI** void **ImageDrawPixel** (**Image** *dst, int posX, int posY, **Color** color)
- **RLAPI** void **ImageDrawPixelV** (**Image** *dst, **Vector2** position, **Color** color)
- **RLAPI** void **ImageDrawLine** (**Image** *dst, int startPosX, int startPosY, int endPosX, int endPosY, **Color** color)
- **RLAPI** void **ImageDrawLineV** (**Image** *dst, **Vector2** start, **Vector2** end, **Color** color)
- **RLAPI** void **ImageDrawLineEx** (**Image** *dst, **Vector2** start, **Vector2** end, int thick, **Color** color)
- **RLAPI** void **ImageDrawCircle** (**Image** *dst, int centerX, int centerY, int radius, **Color** color)
- **RLAPI** void **ImageDrawCircleV** (**Image** *dst, **Vector2** center, int radius, **Color** color)
- **RLAPI** void **ImageDrawCircleLines** (**Image** *dst, int centerX, int centerY, int radius, **Color** color)
- **RLAPI** void **ImageDrawCircleLinesV** (**Image** *dst, **Vector2** center, int radius, **Color** color)
- **RLAPI** void **ImageDrawRectangle** (**Image** *dst, int posX, int posY, int width, int height, **Color** color)
- **RLAPI** void **ImageDrawRectangleV** (**Image** *dst, **Vector2** position, **Vector2** size, **Color** color)
- **RLAPI** void **ImageDrawRectangleRec** (**Image** *dst, **Rectangle** rec, **Color** color)
- **RLAPI** void **ImageDrawRectangleLines** (**Image** *dst, **Rectangle** rec, int thick, **Color** color)
- **RLAPI** void **ImageDrawTriangle** (**Image** *dst, **Vector2** v1, **Vector2** v2, **Vector2** v3, **Color** color)
- **RLAPI** void **ImageDrawTriangleEx** (**Image** *dst, **Vector2** v1, **Vector2** v2, **Vector2** v3, **Color** c1, **Color** c2, **Color** c3)
- **RLAPI** void **ImageDrawTriangleLines** (**Image** *dst, **Vector2** v1, **Vector2** v2, **Vector2** v3, **Color** color)
- **RLAPI** void **ImageDrawTriangleFan** (**Image** *dst, **Vector2** *points, int pointCount, **Color** color)
- **RLAPI** void **ImageDrawTriangleStrip** (**Image** *dst, **Vector2** *points, int pointCount, **Color** color)
- **RLAPI** void **ImageDraw** (**Image** *dst, **Image** src, **Rectangle** srcRec, **Rectangle** dstRec, **Color** tint)
- **RLAPI** void **ImageDrawText** (**Image** *dst, const char *text, int posX, int posY, int fontSize, **Color** color)
- **RLAPI** void **ImageDrawTextEx** (**Image** *dst, **Font** font, const char *text, **Vector2** position, float fontSize, float spacing, **Color** tint)
- **RLAPI Texture2D LoadTexture** (const char *fileName)
- **RLAPI Texture2D LoadTextureFromImage** (**Image** image)
- **RLAPI TextureCubemap LoadTextureCubemap** (**Image** image, int layout)
- **RLAPI RenderTexture2D LoadRenderTexture** (int width, int height)
- **RLAPI bool IsTextureValid** (**Texture2D** texture)
- **RLAPI** void **UnloadTexture** (**Texture2D** texture)
- **RLAPI bool IsRenderTextureValid** (**RenderTexture2D** target)
- **RLAPI** void **UnloadRenderTexture** (**RenderTexture2D** target)
- **RLAPI** void **UpdateTexture** (**Texture2D** texture, const void *pixels)
- **RLAPI** void **UpdateTextureRec** (**Texture2D** texture, **Rectangle** rec, const void *pixels)
- **RLAPI** void **GenTextureMipmaps** (**Texture2D** *texture)
- **RLAPI** void **SetTextureFilter** (**Texture2D** texture, int filter)
- **RLAPI** void **SetTextureWrap** (**Texture2D** texture, int wrap)
- **RLAPI** void **DrawTexture** (**Texture2D** texture, int posX, int posY, **Color** tint)
- **RLAPI** void **DrawTextureV** (**Texture2D** texture, **Vector2** position, **Color** tint)
- **RLAPI** void **DrawTextureEx** (**Texture2D** texture, **Vector2** position, float rotation, float scale, **Color** tint)
- **RLAPI** void **DrawTextureRec** (**Texture2D** texture, **Rectangle** source, **Vector2** position, **Color** tint)
- **RLAPI** void **DrawTexturePro** (**Texture2D** texture, **Rectangle** source, **Rectangle** dest, **Vector2** origin, float rotation, **Color** tint)

- **RLAPI** void **DrawTextureNPatch** (**Texture2D** texture, **NPatchInfo** nPatchInfo, **Rectangle** dest, **Vector2** origin, float rotation, **Color** tint)
- **RLAPI bool ColorIsEqual** (**Color** col1, **Color** col2)
- **RLAPI Color Fade** (**Color** color, float alpha)
- **RLAPI** int **ColorToInt** (**Color** color)
- **RLAPI Vector4 ColorNormalize** (**Color** color)
- **RLAPI Color ColorFromNormalized** (**Vector4** normalized)
- **RLAPI Vector3 ColorToHSV** (**Color** color)
- **RLAPI Color ColorFromHSV** (float hue, float saturation, float value)
- **RLAPI Color ColorTint** (**Color** color, **Color** tint)
- **RLAPI Color ColorBrightness** (**Color** color, float factor)
- **RLAPI Color ColorContrast** (**Color** color, float contrast)
- **RLAPI Color ColorAlpha** (**Color** color, float alpha)
- **RLAPI Color ColorAlphaBlend** (**Color** dst, **Color** src, **Color** tint)
- **RLAPI Color ColorLerp** (**Color** color1, **Color** color2, float factor)
- **RLAPI Color GetColor** (unsigned int hexValue)
- **RLAPI Color GetPixelColor** (void *srcPtr, int format)
- **RLAPI** void **SetPixelColor** (void *dstPtr, **Color** color, int format)
- **RLAPI** int **GetPixelDataSize** (int width, int height, int format)
- **RLAPI Font GetFontDefault** (void)
- **RLAPI Font LoadFont** (const char *fileName)
- **RLAPI Font LoadFontEx** (const char *fileName, int fontSize, int *codepoints, int codepointCount)
- **RLAPI Font LoadFontFromImage** (**Image** image, **Color** key, int firstChar)
- **RLAPI Font LoadFontFromMemory** (const char *fileType, const unsigned char *fileData, int dataSize, int fontSize, int *codepoints, int codepointCount)
- **RLAPI bool IsFontValid** (**Font** font)
- **RLAPI GlyphInfo** * **LoadFontData** (const unsigned char *fileData, int dataSize, int fontSize, int *codepoints, int codepointCount, int type)
- **RLAPI Image GenImageFontAtlas** (const **GlyphInfo** *glyphs, **Rectangle** **glyphRecs, int glyphCount, int fontSize, int padding, int packMethod)
- **RLAPI** void **UnloadFontData** (**GlyphInfo** *glyphs, int glyphCount)
- **RLAPI** void **UnloadFont** (**Font** font)
- **RLAPI bool ExportFontAsCode** (**Font** font, const char *fileName)
- **RLAPI** void **DrawFPS** (int posX, int posY)
- **RLAPI** void **DrawText** (const char *text, int posX, int posY, int fontSize, **Color** color)
- **RLAPI** void **DrawTextEx** (**Font** font, const char *text, **Vector2** position, float fontSize, float spacing, **Color** tint)
- **RLAPI** void **DrawTextPro** (**Font** font, const char *text, **Vector2** position, **Vector2** origin, float rotation, float fontSize, float spacing, **Color** tint)
- **RLAPI** void **DrawTextCodepoint** (**Font** font, int codepoint, **Vector2** position, float fontSize, **Color** tint)
- **RLAPI** void **DrawTextCodepoints** (**Font** font, const int *codepoints, int codepointCount, **Vector2** position, float fontSize, float spacing, **Color** tint)
- **RLAPI** void **SetTextLineSpacing** (int spacing)
- **RLAPI** int **MeasureText** (const char *text, int fontSize)
- **RLAPI Vector2 MeasureTextEx** (**Font** font, const char *text, float fontSize, float spacing)
- **RLAPI** int **GetGlyphIndex** (**Font** font, int codepoint)
- **RLAPI GlyphInfo GetGlyphInfo** (**Font** font, int codepoint)
- **RLAPI Rectangle GetGlyphAtlasRec** (**Font** font, int codepoint)
- **RLAPI** char * **LoadUTF8** (const int *codepoints, int length)
- **RLAPI** void **UnloadUTF8** (char *text)
- **RLAPI** int * **LoadCodepoints** (const char *text, int *count)
- **RLAPI** void **UnloadCodepoints** (int *codepoints)
- **RLAPI** int **GetCodepointCount** (const char *text)
- **RLAPI** int **GetCodepoint** (const char *text, int *codepointSize)
- **RLAPI** int **GetCodepointNext** (const char *text, int *codepointSize)
- **RLAPI** int **GetCodepointPrevious** (const char *text, int *codepointSize)
- **RLAPI** const char * **CodepointToUTF8** (int codepoint, int *utf8Size)
- **RLAPI** int **TextCopy** (char *dst, const char *src)

- **RLAPI bool TextIsEqual** (const char *text1, const char *text2)
- **RLAPI** unsigned int **TextLength** (const char *text)
- **RLAPI** const char * **TextFormat** (const char *text,...)
- **RLAPI** const char * **TextSubtext** (const char *text, int position, int length)
- **RLAPI** char * **TextReplace** (const char *text, const char *replace, const char *by)
- **RLAPI** char * **TextInsert** (const char *text, const char *insert, int position)
- **RLAPI** const char * **TextJoin** (const char **textList, int count, const char *delimiter)
- **RLAPI** const char ** **TextSplit** (const char *text, char delimiter, int *count)
- **RLAPI** void **TextAppend** (char *text, const char *append, int *position)
- **RLAPI** int **TextFindIndex** (const char *text, const char *find)
- **RLAPI** const char * **TextToUpper** (const char *text)
- **RLAPI** const char * **TextToLower** (const char *text)
- **RLAPI** const char * **TextToPascal** (const char *text)
- **RLAPI** const char * **TextToSnake** (const char *text)
- **RLAPI** const char * **TextToCamel** (const char *text)
- **RLAPI** int **TextToInteger** (const char *text)
- **RLAPI** float **TextToFloat** (const char *text)
- **RLAPI** void **DrawLine3D** (**Vector3** startPos, **Vector3** endPos, **Color** color)
- **RLAPI** void **DrawPoint3D** (**Vector3** position, **Color** color)
- **RLAPI** void **DrawCircle3D** (**Vector3** center, float radius, **Vector3** rotationAxis, float rotationAngle, **Color** color)
- **RLAPI** void **DrawTriangle3D** (**Vector3** v1, **Vector3** v2, **Vector3** v3, **Color** color)
- **RLAPI** void **DrawTriangleStrip3D** (const **Vector3** *points, int pointCount, **Color** color)
- **RLAPI** void **DrawCube** (**Vector3** position, float width, float height, float length, **Color** color)
- **RLAPI** void **DrawCubeV** (**Vector3** position, **Vector3** size, **Color** color)
- **RLAPI** void **DrawCubeWires** (**Vector3** position, float width, float height, float length, **Color** color)
- **RLAPI** void **DrawCubeWiresV** (**Vector3** position, **Vector3** size, **Color** color)
- **RLAPI** void **DrawSphere** (**Vector3** centerPos, float radius, **Color** color)
- **RLAPI** void **DrawSphereEx** (**Vector3** centerPos, float radius, int rings, int slices, **Color** color)
- **RLAPI** void **DrawSphereWires** (**Vector3** centerPos, float radius, int rings, int slices, **Color** color)
- **RLAPI** void **DrawCylinder** (**Vector3** position, float radiusTop, float radiusBottom, float height, int slices, **Color** color)
- **RLAPI** void **DrawCylinderEx** (**Vector3** startPos, **Vector3** endPos, float startRadius, float endRadius, int sides, **Color** color)
- **RLAPI** void **DrawCylinderWires** (**Vector3** position, float radiusTop, float radiusBottom, float height, int slices, **Color** color)
- **RLAPI** void **DrawCylinderWiresEx** (**Vector3** startPos, **Vector3** endPos, float startRadius, float endRadius, int sides, **Color** color)
- **RLAPI** void **DrawCapsule** (**Vector3** startPos, **Vector3** endPos, float radius, int slices, int rings, **Color** color)
- **RLAPI** void **DrawCapsuleWires** (**Vector3** startPos, **Vector3** endPos, float radius, int slices, int rings, **Color** color)
- **RLAPI** void **DrawPlane** (**Vector3** centerPos, **Vector2** size, **Color** color)
- **RLAPI** void **DrawRay** (**Ray** ray, **Color** color)
- **RLAPI** void **DrawGrid** (int slices, float spacing)
- **RLAPI Model LoadModel** (const char *fileName)
- **RLAPI Model LoadModelFromMesh** (**Mesh** mesh)
- **RLAPI bool IsModelValid** (**Model** model)
- **RLAPI** void **UnloadModel** (**Model** model)
- **RLAPI BoundingBox GetModelBoundingBox** (**Model** model)
- **RLAPI** void **DrawModel** (**Model** model, **Vector3** position, float scale, **Color** tint)
- **RLAPI** void **DrawModelEx** (**Model** model, **Vector3** position, **Vector3** rotationAxis, float rotationAngle, **Vector3** scale, **Color** tint)
- **RLAPI** void **DrawModelWires** (**Model** model, **Vector3** position, float scale, **Color** tint)
- **RLAPI** void **DrawModelWiresEx** (**Model** model, **Vector3** position, **Vector3** rotationAxis, float rotationAngle, **Vector3** scale, **Color** tint)
- **RLAPI** void **DrawModelPoints** (**Model** model, **Vector3** position, float scale, **Color** tint)

- **RLAPI** void **DrawModelPointsEx** (**Model** model, **Vector3** position, **Vector3** rotationAxis, float rotationAngle, **Vector3** scale, **Color** tint)
- **RLAPI** void **DrawBoundingBox** (**BoundingBox** box, **Color** color)
- **RLAPI** void **DrawBillboard** (**Camera** camera, **Texture2D** texture, **Vector3** position, float scale, **Color** tint)
- **RLAPI** void **DrawBillboardRec** (**Camera** camera, **Texture2D** texture, **Rectangle** source, **Vector3** position, **Vector2** size, **Color** tint)
- **RLAPI** void **DrawBillboardPro** (**Camera** camera, **Texture2D** texture, **Rectangle** source, **Vector3** position, **Vector3** up, **Vector2** size, **Vector2** origin, float rotation, **Color** tint)
- **RLAPI** void **UploadMesh** (**Mesh** *mesh, **bool** dynamic)
- **RLAPI** void **UpdateMeshBuffer** (**Mesh** mesh, int index, const void *data, int dataSize, int offset)
- **RLAPI** void **UnloadMesh** (**Mesh** mesh)
- **RLAPI** void **DrawMesh** (**Mesh** mesh, **Material** material, **Matrix** transform)
- **RLAPI** void **DrawMeshInstanced** (**Mesh** mesh, **Material** material, const **Matrix** *transforms, int instances)
- **RLAPI BoundingBox GetMeshBoundingBox** (**Mesh** mesh)
- **RLAPI** void **GenMeshTangents** (**Mesh** *mesh)
- **RLAPI bool ExportMesh** (**Mesh** mesh, const char *fileName)
- **RLAPI bool ExportMeshAsCode** (**Mesh** mesh, const char *fileName)
- **RLAPI Mesh GenMeshPoly** (int sides, float radius)
- **RLAPI Mesh GenMeshPlane** (float width, float length, int resX, int resZ)
- **RLAPI Mesh GenMeshCube** (float width, float height, float length)
- **RLAPI Mesh GenMeshSphere** (float radius, int rings, int slices)
- **RLAPI Mesh GenMeshHemiSphere** (float radius, int rings, int slices)
- **RLAPI Mesh GenMeshCylinder** (float radius, float height, int slices)
- **RLAPI Mesh GenMeshCone** (float radius, float height, int slices)
- **RLAPI Mesh GenMeshTorus** (float radius, float size, int radSeg, int sides)
- **RLAPI Mesh GenMeshKnot** (float radius, float size, int radSeg, int sides)
- **RLAPI Mesh GenMeshHeightmap** (**Image** heightmap, **Vector3** size)
- **RLAPI Mesh GenMeshCubicmap** (**Image** cubicmap, **Vector3** cubeSize)
- **RLAPI Material** * **LoadMaterials** (const char *fileName, int *materialCount)
- **RLAPI Material LoadMaterialDefault** (void)
- **RLAPI bool IsMaterialValid** (**Material** material)
- **RLAPI** void **UnloadMaterial** (**Material** material)
- **RLAPI** void **SetMaterialTexture** (**Material** *material, int mapType, **Texture2D** texture)
- **RLAPI** void **SetModelMeshMaterial** (**Model** *model, int meshId, int materialId)
- **RLAPI ModelAnimation** * **LoadModelAnimations** (const char *fileName, int *animCount)
- **RLAPI** void **UpdateModelAnimation** (**Model** model, **ModelAnimation** anim, int frame)
- **RLAPI** void **UpdateModelAnimationBones** (**Model** model, **ModelAnimation** anim, int frame)
- **RLAPI** void **UnloadModelAnimation** (**ModelAnimation** anim)
- **RLAPI** void **UnloadModelAnimations** (**ModelAnimation** *animations, int animCount)
- **RLAPI bool IsModelAnimationValid** (**Model** model, **ModelAnimation** anim)
- **RLAPI bool CheckCollisionSpheres** (**Vector3** center1, float radius1, **Vector3** center2, float radius2)
- **RLAPI bool CheckCollisionBoxes** (**BoundingBox** box1, **BoundingBox** box2)
- **RLAPI bool CheckCollisionBoxSphere** (**BoundingBox** box, **Vector3** center, float radius)
- **RLAPI RayCollision GetRayCollisionSphere** (**Ray** ray, **Vector3** center, float radius)
- **RLAPI RayCollision GetRayCollisionBox** (**Ray** ray, **BoundingBox** box)
- **RLAPI RayCollision GetRayCollisionMesh** (**Ray** ray, **Mesh** mesh, **Matrix** transform)
- **RLAPI RayCollision GetRayCollisionTriangle** (**Ray** ray, **Vector3** p1, **Vector3** p2, **Vector3** p3)
- **RLAPI RayCollision GetRayCollisionQuad** (**Ray** ray, **Vector3** p1, **Vector3** p2, **Vector3** p3, **Vector3** p4)
- **RLAPI** void **InitAudioDevice** (void)
- **RLAPI** void **CloseAudioDevice** (void)
- **RLAPI bool IsAudioDeviceReady** (void)
- **RLAPI** void **SetMasterVolume** (float volume)
- **RLAPI** float **GetMasterVolume** (void)
- **RLAPI Wave LoadWave** (const char *fileName)
- **RLAPI Wave LoadWaveFromMemory** (const char *fileType, const unsigned char *fileData, int dataSize)

- **RLAPI bool IsWaveValid** (**Wave** wave)
- **RLAPI Sound LoadSound** (const char *fileName)
- **RLAPI Sound LoadSoundFromWave** (**Wave** wave)
- **RLAPI Sound LoadSoundAlias** (**Sound** source)
- **RLAPI bool IsSoundValid** (**Sound** sound)
- **RLAPI** void **UpdateSound** (**Sound** sound, const void *data, int sampleCount)
- **RLAPI** void **UnloadWave** (**Wave** wave)
- **RLAPI** void **UnloadSound** (**Sound** sound)
- **RLAPI** void **UnloadSoundAlias** (**Sound** alias)
- **RLAPI bool ExportWave** (**Wave** wave, const char *fileName)
- **RLAPI bool ExportWaveAsCode** (**Wave** wave, const char *fileName)
- **RLAPI** void **PlaySound** (**Sound** sound)
- **RLAPI** void **StopSound** (**Sound** sound)
- **RLAPI** void **PauseSound** (**Sound** sound)
- **RLAPI** void **ResumeSound** (**Sound** sound)
- **RLAPI bool IsSoundPlaying** (**Sound** sound)
- **RLAPI** void **SetSoundVolume** (**Sound** sound, float volume)
- **RLAPI** void **SetSoundPitch** (**Sound** sound, float pitch)
- **RLAPI** void **SetSoundPan** (**Sound** sound, float pan)
- **RLAPI Wave WaveCopy** (**Wave** wave)
- **RLAPI** void **WaveCrop** (**Wave** *wave, int initFrame, int finalFrame)
- **RLAPI** void **WaveFormat** (**Wave** *wave, int sampleRate, int sampleSize, int channels)
- **RLAPI** float * **LoadWaveSamples** (**Wave** wave)
- **RLAPI** void **UnloadWaveSamples** (float *samples)
- **RLAPI Music LoadMusicStream** (const char *fileName)
- **RLAPI Music LoadMusicStreamFromMemory** (const char *fileType, const unsigned char *data, int dataSize)
- **RLAPI bool IsMusicValid** (**Music** music)
- **RLAPI** void **UnloadMusicStream** (**Music** music)
- **RLAPI** void **PlayMusicStream** (**Music** music)
- **RLAPI bool IsMusicStreamPlaying** (**Music** music)
- **RLAPI** void **UpdateMusicStream** (**Music** music)
- **RLAPI** void **StopMusicStream** (**Music** music)
- **RLAPI** void **PauseMusicStream** (**Music** music)
- **RLAPI** void **ResumeMusicStream** (**Music** music)
- **RLAPI** void **SeekMusicStream** (**Music** music, float position)
- **RLAPI** void **SetMusicVolume** (**Music** music, float volume)
- **RLAPI** void **SetMusicPitch** (**Music** music, float pitch)
- **RLAPI** void **SetMusicPan** (**Music** music, float pan)
- **RLAPI** float **GetMusicTimeLength** (**Music** music)
- **RLAPI** float **GetMusicTimePlayed** (**Music** music)
- **RLAPI AudioStream LoadAudioStream** (unsigned int sampleRate, unsigned int sampleSize, unsigned int channels)
- **RLAPI bool IsAudioStreamValid** (**AudioStream** stream)
- **RLAPI** void **UnloadAudioStream** (**AudioStream** stream)
- **RLAPI** void **UpdateAudioStream** (**AudioStream** stream, const void *data, int frameCount)
- **RLAPI bool IsAudioStreamProcessed** (**AudioStream** stream)
- **RLAPI** void **PlayAudioStream** (**AudioStream** stream)
- **RLAPI** void **PauseAudioStream** (**AudioStream** stream)
- **RLAPI** void **ResumeAudioStream** (**AudioStream** stream)
- **RLAPI bool IsAudioStreamPlaying** (**AudioStream** stream)
- **RLAPI** void **StopAudioStream** (**AudioStream** stream)
- **RLAPI** void **SetAudioStreamVolume** (**AudioStream** stream, float volume)
- **RLAPI** void **SetAudioStreamPitch** (**AudioStream** stream, float pitch)
- **RLAPI** void **SetAudioStreamPan** (**AudioStream** stream, float pan)
- **RLAPI** void **SetAudioStreamBufferSizeDefault** (int size)
- **RLAPI** void **SetAudioStreamCallback** (**AudioStream** stream, **AudioCallback** callback)
- **RLAPI** void **AttachAudioStreamProcessor** (**AudioStream** stream, **AudioCallback** processor)
- **RLAPI** void **DetachAudioStreamProcessor** (**AudioStream** stream, **AudioCallback** processor)
- **RLAPI** void **AttachAudioMixedProcessor** (**AudioCallback** processor)

- **RLAPI** void **DetachAudioMixedProcessor** (**AudioCallback** processor)

Definições e macros

#define BEIGE   CLITERAL(Color){ 211, 176, 131, 255 }

#define BLACK   CLITERAL(Color){ 0, 0, 0, 255 }

#define BLANK   CLITERAL(Color){ 0, 0, 0, 0 }

#define BLUE   CLITERAL(Color){ 0, 121, 241, 255 }

#define BROWN   CLITERAL(Color){ 127, 106, 79, 255 }

#define CLITERAL(type)

Valor:

```
(type)
```

```
#define DARKBLUE   CLITERAL(Color){ 0, 82, 172, 255 }

#define DARKBROWN   CLITERAL(Color){ 76, 63, 47, 255 }

#define DARKGRAY   CLITERAL(Color){ 80, 80, 80, 255 }

#define DARKGREEN   CLITERAL(Color){ 0, 117, 44, 255 }

#define DARKPURPLE   CLITERAL(Color){ 112, 31, 126, 255 }

#define DEG2RAD   (PI/180.0f)

#define GetMouseRay   GetScreenToWorldRay

#define GOLD   CLITERAL(Color){ 255, 203, 0, 255 }

#define GRAY   CLITERAL(Color){ 130, 130, 130, 255 }

#define GREEN   CLITERAL(Color){ 0, 228, 48, 255 }

#define LIGHTGRAY   CLITERAL(Color){ 200, 200, 200, 255 }

#define LIME   CLITERAL(Color){ 0, 158, 47, 255 }

#define MAGENTA   CLITERAL(Color){ 255, 0, 255, 255 }

#define MAROON   CLITERAL(Color){ 190, 33, 55, 255 }

#define MATERIAL_MAP_DIFFUSE   MATERIAL_MAP_ALBEDO

#define MATERIAL_MAP_SPECULAR   MATERIAL_MAP_METALNESS

#define MOUSE_LEFT_BUTTON   MOUSE_BUTTON_LEFT

#define MOUSE_MIDDLE_BUTTON   MOUSE_BUTTON_MIDDLE

#define MOUSE_RIGHT_BUTTON   MOUSE_BUTTON_RIGHT

#define ORANGE   CLITERAL(Color){ 255, 161, 0, 255 }

#define PI   3.14159265358979323846f

#define PINK   CLITERAL(Color){ 255, 109, 194, 255 }

#define PURPLE   CLITERAL(Color){ 200, 122, 255, 255 }

#define RAD2DEG   (180.0f/PI)

#define RAYLIB_VERSION   "5.5"

#define RAYLIB_VERSION_MAJOR   5
```

#define RAYLIB_VERSION_MINOR   5

#define RAYLIB_VERSION_PATCH   0

#define RAYWHITE   CLITERAL(Color){ 245, 245, 245, 255 }

#define RED   CLITERAL(Color){ 230, 41, 55, 255 }

#define RL_BOOL_TYPE

#define RL_CALLOC(n,   sz)

Valor:

```
calloc(n,sz)
```

#define RL_COLOR_TYPE

#define RL_FREE(ptr)

Valor:

```
free(ptr)
```

#define RL_MALLOC(sz)

Valor:

```
malloc(sz)
```

#define RL_MATRIX_TYPE

#define RL_QUATERNION_TYPE

#define RL_REALLOC(ptr,   sz)

Valor:

```
realloc(ptr,sz)
```

#define RL_RECTANGLE_TYPE

#define RL_VECTOR2_TYPE

#define RL_VECTOR3_TYPE

#define RL_VECTOR4_TYPE

#define RLAPI

#define SHADER_LOC_MAP_DIFFUSE   SHADER_LOC_MAP_ALBEDO

#define SHADER_LOC_MAP_SPECULAR   SHADER_LOC_MAP_METALNESS

#define SKYBLUE   CLITERAL(Color){ 102, 191, 255, 255 }

#define VIOLET   CLITERAL(Color){ 135, 60, 190, 255 }

#define WHITE   CLITERAL(Color){ 255, 255, 255, 255 }

#define YELLOW   CLITERAL(Color){ 253, 249, 0, 255 }

Definições dos tipos

typedef void(* AudioCallback) (void *bufferData, unsigned int frames)

typedef struct AudioStream AudioStream

typedef struct AutomationEvent AutomationEvent

typedef struct AutomationEventList AutomationEventList

typedef struct BoneInfo BoneInfo

typedef enum bool bool

typedef struct BoundingBox BoundingBox

typedef Camera3D Camera

typedef struct Camera2D Camera2D

typedef struct Camera3D Camera3D

typedef struct Color Color

typedef struct FilePathList FilePathList

typedef struct Font Font

typedef struct GlyphInfo GlyphInfo

typedef struct Image Image

typedef unsigned char *(* LoadFileDataCallback) (const char *fileName, int *dataSize)

typedef char *(* LoadFileTextCallback) (const char *fileName)

typedef struct Material Material

typedef struct MaterialMap MaterialMap

typedef struct Matrix Matrix

typedef struct Mesh Mesh

typedef struct Model Model

typedef struct ModelAnimation ModelAnimation

typedef struct Music Music

typedef struct NPatchInfo NPatchInfo

typedef Vector4 Quaternion

typedef struct rAudioBuffer rAudioBuffer

typedef struct rAudioProcessor rAudioProcessor

typedef struct Ray Ray

typedef struct RayCollision RayCollision

typedef struct Rectangle Rectangle

typedef struct RenderTexture RenderTexture

typedef RenderTexture RenderTexture2D

typedef bool(* SaveFileDataCallback) (const char *fileName, void *data, int dataSize)

typedef bool(* SaveFileTextCallback) (const char *fileName, char *text)

typedef struct Shader Shader

typedef struct Sound Sound

typedef struct Texture Texture

typedef Texture Texture2D

typedef Texture TextureCubemap

typedef void(* TraceLogCallback) (int logLevel, const char *text, va_list args)

typedef struct Transform Transform

typedef struct Vector2 Vector2

typedef struct Vector3 Vector3

typedef struct Vector4 Vector4

typedef struct VrDeviceInfo VrDeviceInfo

typedef struct VrStereoConfig VrStereoConfig

typedef struct Wave Wave

Enumerações

enum BlendMode

Enumeradores:

BLEND_ALPHA	
BLEND_ADDITIVE	
BLEND_MULTIPLIED	
BLEND_ADD_COLORS	
BLEND_SUBTRACT_COLORS	
BLEND_ALPHA_PREMULTIPLY	
BLEND_CUSTOM	
BLEND_CUSTOM_SEPARATE	

```
            {
    BLEND_ALPHA = 0,                // Blend textures considering alpha (default)
    BLEND_ADDITIVE,                 // Blend textures adding colors
    BLEND_MULTIPLIED,               // Blend textures multiplying colors
    BLEND_ADD_COLORS,               // Blend textures adding colors (alternative)
    BLEND_SUBTRACT_COLORS,          // Blend textures subtracting colors (alternative)
    BLEND_ALPHA_PREMULTIPLY,        // Blend premultiplied textures considering alpha
    BLEND_CUSTOM,                   // Blend textures using custom src/dst factors (use rlSetBlendFactors())
    BLEND_CUSTOM_SEPARATE           // Blend textures using custom rgb/alpha separate src/dst factors (use rlSetBlendFactorsSeparate())
} BlendMode;
```

enum bool

Enumeradores:

false	
true	

```
{ false = 0, true = !false } bool;
```

enum CameraMode

Enumeradores:

CAMERA_CUSTOM	
CAMERA_FREE	
CAMERA_ORB	

ITAL	
CAMERA_FIRST_PERSON	
CAMERA_THIRD_PERSON	

```
          {
    CAMERA_CUSTOM = 0,              // Camera custom, controlled by user
(UpdateCamera() does nothing)
    CAMERA_FREE,                    // Camera free mode
    CAMERA_ORBITAL,                 // Camera orbital, around target, zoom
supported
    CAMERA_FIRST_PERSON,            // Camera first person
    CAMERA_THIRD_PERSON             // Camera third person
} CameraMode;
```

enum CameraProjection

Enumeradores:

CAMERA_PERSPECTIVE	
CAMERA_ORTHOGRAPHIC	

```
          {
    CAMERA_PERSPECTIVE = 0,         // Perspective projection
    CAMERA_ORTHOGRAPHIC             // Orthographic projection
} CameraProjection;
```

enum ConfigFlags

Enumeradores:

FLAG_VSYNC_HINT	
FLAG_FULLSCREEN_MODE	
FLAG_WINDOW_RESIZABLE	
FLAG_WINDOW_UNDECORATED	
FLAG_WINDOW_HIDDEN	
FLAG_WINDOW_MINIMIZED	
FLAG_WINDOW_MAXIMIZED	
FLAG_WINDOW_UNFOCUSED	

FLAG_WINDOW_TOPMOST	
FLAG_WINDOW_ALWAYS_RUN	
FLAG_WINDOW_TRANSPARENT	
FLAG_WINDOW_HIGHDPI	
FLAG_WINDOW_MOUSE_PASSTHROUGH	
FLAG_BORDERLESS_WINDOWED_MODE	
FLAG_MSAA_4X_HINT	
FLAG_INTERLACED_HINT	

```
                 {
        FLAG_VSYNC_HINT         = 0x00000040,   // Set to try enabling V-Sync on GPU
        FLAG_FULLSCREEN_MODE    = 0x00000002,   // Set to run program in fullscreen
        FLAG_WINDOW_RESIZABLE   = 0x00000004,   // Set to allow resizable window
        FLAG_WINDOW_UNDECORATED = 0x00000008,   // Set to disable window decoration (frame and buttons)
        FLAG_WINDOW_HIDDEN      = 0x00000080,   // Set to hide window
        FLAG_WINDOW_MINIMIZED   = 0x00000200,   // Set to minimize window (iconify)
        FLAG_WINDOW_MAXIMIZED   = 0x00000400,   // Set to maximize window (expanded to monitor)
        FLAG_WINDOW_UNFOCUSED   = 0x00000800,   // Set to window non focused
        FLAG_WINDOW_TOPMOST     = 0x00001000,   // Set to window always on top
        FLAG_WINDOW_ALWAYS_RUN  = 0x00000100,   // Set to allow windows running while minimized
        FLAG_WINDOW_TRANSPARENT = 0x00000010,   // Set to allow transparent framebuffer
        FLAG_WINDOW_HIGHDPI     = 0x00002000,   // Set to support HighDPI
        FLAG_WINDOW_MOUSE_PASSTHROUGH = 0x00004000, // Set to support mouse passthrough, only supported when FLAG_WINDOW_UNDECORATED
        FLAG_BORDERLESS_WINDOWED_MODE = 0x00008000, // Set to run program in borderless windowed mode
        FLAG_MSAA_4X_HINT       = 0x00000020,   // Set to try enabling MSAA 4X
        FLAG_INTERLACED_HINT    = 0x00010000    // Set to try enabling interlaced video format (for V3D)
} ConfigFlags;
```

enum CubemapLayout

Enumeradores:

CUBEMAP_LAYOUT_AUTO_DETECT	
CUBEMAP_LAYOUT_LINE_VERTICAL	
CUBEMAP_LA	

YOUT_LINE_HORIZONTAL	
CUBEMAP_LAYOUT_CROSS_THREE_BY_FOUR	
CUBEMAP_LAYOUT_CROSS_FOUR_BY_THREE	

```
            {
    CUBEMAP_LAYOUT_AUTO_DETECT = 0,         // Automatically detect layout type
    CUBEMAP_LAYOUT_LINE_VERTICAL,           // Layout is defined by a vertical line with faces
    CUBEMAP_LAYOUT_LINE_HORIZONTAL,         // Layout is defined by a horizontal line with faces
    CUBEMAP_LAYOUT_CROSS_THREE_BY_FOUR,     // Layout is defined by a 3x4 cross with cubemap faces
    CUBEMAP_LAYOUT_CROSS_FOUR_BY_THREE     // Layout is defined by a 4x3 cross with cubemap faces
} CubemapLayout;
```

enum FontType

Enumeradores:

FONT_DEFAULT	
FONT_BITMAP	
FONT_SDF	

```
            {
    FONT_DEFAULT = 0,               // Default font generation, anti-aliased
    FONT_BITMAP,                    // Bitmap font generation, no anti-aliasing
    FONT_SDF                        // SDF font generation, requires external shader
} FontType;
```

enum GamepadAxis

Enumeradores:

GAMEPAD_AXIS_LEFT_X	
GAMEPAD_AXIS_LEFT_Y	
GAMEPAD_AXIS_RIGHT_X	
GAMEPAD_AXIS_RIGHT_Y	
GAMEPAD_AXIS_LEFT_TRIGGER	
GAMEPAD_AX	

IS_RIGHT_TRIGGER	

```
            {
    GAMEPAD_AXIS_LEFT_X         = 0,     // Gamepad left stick X axis
    GAMEPAD_AXIS_LEFT_Y         = 1,     // Gamepad left stick Y axis
    GAMEPAD_AXIS_RIGHT_X        = 2,     // Gamepad right stick X axis
    GAMEPAD_AXIS_RIGHT_Y        = 3,     // Gamepad right stick Y axis
    GAMEPAD_AXIS_LEFT_TRIGGER   = 4,     // Gamepad back trigger left, pressure level: [1..-1]
    GAMEPAD_AXIS_RIGHT_TRIGGER = 5       // Gamepad back trigger right, pressure level: [1..-1]
} GamepadAxis;
```

enum GamepadButton

Enumeradores:

GAMEPAD_BUTTON_UNKNOWN	
GAMEPAD_BUTTON_LEFT_FACE_UP	
GAMEPAD_BUTTON_LEFT_FACE_RIGHT	
GAMEPAD_BUTTON_LEFT_FACE_DOWN	
GAMEPAD_BUTTON_LEFT_FACE_LEFT	
GAMEPAD_BUTTON_RIGHT_FACE_UP	
GAMEPAD_BUTTON_RIGHT_FACE_RIGHT	
GAMEPAD_BUTTON_RIGHT_FACE_DOWN	
GAMEPAD_BUTTON_RIGHT_FACE_LEFT	
GAMEPAD_BUTTON_LEFT_TRIGGER_1	
GAMEPAD_BUTTON_LEFT_TRIGGER_2	
GAMEPAD_BUTTON_RIGHT_	

TRIGGER_1	
GAMEPAD_BUTTON_RIGHT_TRIGGER_2	
GAMEPAD_BUTTON_MIDDLE_LEFT	
GAMEPAD_BUTTON_MIDDLE	
GAMEPAD_BUTTON_MIDDLE_RIGHT	
GAMEPAD_BUTTON_LEFT_THUMB	
GAMEPAD_BUTTON_RIGHT_THUMB	

```
             {
     GAMEPAD_BUTTON_UNKNOWN = 0,          // Unknown button, just for error checking
     GAMEPAD_BUTTON_LEFT_FACE_UP,          // Gamepad left DPAD up button
     GAMEPAD_BUTTON_LEFT_FACE_RIGHT,       // Gamepad left DPAD right button
     GAMEPAD_BUTTON_LEFT_FACE_DOWN,        // Gamepad left DPAD down button
     GAMEPAD_BUTTON_LEFT_FACE_LEFT,        // Gamepad left DPAD left button
     GAMEPAD_BUTTON_RIGHT_FACE_UP,         // Gamepad right button up (i.e. PS3:
Triangle, Xbox: Y)
     GAMEPAD_BUTTON_RIGHT_FACE_RIGHT,      // Gamepad right button right (i.e. PS3:
Circle, Xbox: B)
     GAMEPAD_BUTTON_RIGHT_FACE_DOWN,       // Gamepad right button down (i.e. PS3:
Cross, Xbox: A)
     GAMEPAD_BUTTON_RIGHT_FACE_LEFT,       // Gamepad right button left (i.e. PS3:
Square, Xbox: X)
     GAMEPAD_BUTTON_LEFT_TRIGGER_1,        // Gamepad top/back trigger left (first),
it could be a trailing button
     GAMEPAD_BUTTON_LEFT_TRIGGER_2,        // Gamepad top/back trigger left
(second), it could be a trailing button
     GAMEPAD_BUTTON_RIGHT_TRIGGER_1,       // Gamepad top/back trigger right
(first), it could be a trailing button
     GAMEPAD_BUTTON_RIGHT_TRIGGER_2,       // Gamepad top/back trigger right
(second), it could be a trailing button
     GAMEPAD_BUTTON_MIDDLE_LEFT,           // Gamepad center buttons, left one (i.e.
PS3: Select)
     GAMEPAD_BUTTON_MIDDLE,                // Gamepad center buttons, middle one
(i.e. PS3: PS, Xbox: XBOX)
     GAMEPAD_BUTTON_MIDDLE_RIGHT,          // Gamepad center buttons, right one
(i.e. PS3: Start)
     GAMEPAD_BUTTON_LEFT_THUMB,            // Gamepad joystick pressed button left
     GAMEPAD_BUTTON_RIGHT_THUMB            // Gamepad joystick pressed button right
 } GamepadButton;
```

enum Gesture

Enumeradores:

GESTURE_NONE	
GESTURE_TAP	
GESTURE_DO	

UBLETAP	
GESTURE_HOLD	
GESTURE_DRAG	
GESTURE_SWIPE_RIGHT	
GESTURE_SWIPE_LEFT	
GESTURE_SWIPE_UP	
GESTURE_SWIPE_DOWN	
GESTURE_PINCH_IN	
GESTURE_PINCH_OUT	

```
               {
    GESTURE_NONE        = 0,        // No gesture
    GESTURE_TAP         = 1,        // Tap gesture
    GESTURE_DOUBLETAP   = 2,        // Double tap gesture
    GESTURE_HOLD        = 4,        // Hold gesture
    GESTURE_DRAG        = 8,        // Drag gesture
    GESTURE_SWIPE_RIGHT = 16,       // Swipe right gesture
    GESTURE_SWIPE_LEFT  = 32,       // Swipe left gesture
    GESTURE_SWIPE_UP    = 64,       // Swipe up gesture
    GESTURE_SWIPE_DOWN  = 128,      // Swipe down gesture
    GESTURE_PINCH_IN    = 256,      // Pinch in gesture
    GESTURE_PINCH_OUT   = 512       // Pinch out gesture
} Gesture;
```

enum KeyboardKey

Enumeradores:

KEY_NULL	
KEY_APOSTROPHE	
KEY_COMMA	
KEY_MINUS	
KEY_PERIOD	
KEY_SLASH	
KEY_ZERO	
KEY_ONE	
KEY_TWO	
KEY_THREE	
KEY_FOUR	
KEY_FIVE	
KEY_SIX	

KEY_SEVEN	
KEY_EIGHT	
KEY_NINE	
KEY_SEMICOLON	
KEY_EQUAL	
KEY_A	
KEY_B	
KEY_C	
KEY_D	
KEY_E	
KEY_F	
KEY_G	
KEY_H	
KEY_I	
KEY_J	
KEY_K	
KEY_L	
KEY_M	
KEY_N	
KEY_O	
KEY_P	
KEY_Q	
KEY_R	
KEY_S	
KEY_T	
KEY_U	
KEY_V	
KEY_W	
KEY_X	
KEY_Y	
KEY_Z	
KEY_LEFT_BRACKET	
KEY_BACKSLASH	
KEY_RIGHT_BRACKET	
KEY_GRAVE	
KEY_SPACE	

KEY_ESCAPE	
KEY_ENTER	
KEY_TAB	
KEY_BACKSPACE	
KEY_INSERT	
KEY_DELETE	
KEY_RIGHT	
KEY_LEFT	
KEY_DOWN	
KEY_UP	
KEY_PAGE_UP	
KEY_PAGE_DOWN	
KEY_HOME	
KEY_END	
KEY_CAPS_LOCK	
KEY_SCROLL_LOCK	
KEY_NUM_LOCK	
KEY_PRINT_SCREEN	
KEY_PAUSE	
KEY_F1	
KEY_F2	
KEY_F3	
KEY_F4	
KEY_F5	
KEY_F6	
KEY_F7	
KEY_F8	
KEY_F9	
KEY_F10	
KEY_F11	
KEY_F12	
KEY_LEFT_SHIFT	
KEY_LEFT_CONTROL	

KEY_LEFT_ALT	
KEY_LEFT_SUPER	
KEY_RIGHT_SHIFT	
KEY_RIGHT_CONTROL	
KEY_RIGHT_ALT	
KEY_RIGHT_SUPER	
KEY_KB_MENU	
KEY_KP_0	
KEY_KP_1	
KEY_KP_2	
KEY_KP_3	
KEY_KP_4	
KEY_KP_5	
KEY_KP_6	
KEY_KP_7	
KEY_KP_8	
KEY_KP_9	
KEY_KP_DECIMAL	
KEY_KP_DIVIDE	
KEY_KP_MULTIPLY	
KEY_KP_SUBTRACT	
KEY_KP_ADD	
KEY_KP_ENTER	
KEY_KP_EQUAL	
KEY_BACK	
KEY_MENU	
KEY_VOLUME_UP	
KEY_VOLUME_DOWN	

```
        {
    KEY_NULL            = 0,        // Key: NULL, used for no key pressed
    // Alphanumeric keys
    KEY_APOSTROPHE      = 39,       // Key: '
    KEY_COMMA           = 44,       // Key: ,
    KEY_MINUS           = 45,       // Key: -
    KEY_PERIOD          = 46,       // Key: .
    KEY_SLASH           = 47,       // Key: /
    KEY_ZERO            = 48,       // Key: 0
    KEY_ONE             = 49,       // Key: 1
    KEY_TWO             = 50,       // Key: 2
    KEY_THREE           = 51,       // Key: 3
    KEY_FOUR            = 52,       // Key: 4
    KEY_FIVE            = 53,       // Key: 5
    KEY_SIX             = 54,       // Key: 6
    KEY_SEVEN           = 55,       // Key: 7
    KEY_EIGHT           = 56,       // Key: 8
    KEY_NINE            = 57,       // Key: 9
    KEY_SEMICOLON       = 59,       // Key: ;
    KEY_EQUAL           = 61,       // Key: =
    KEY_A               = 65,       // Key: A | a
    KEY_B               = 66,       // Key: B | b
    KEY_C               = 67,       // Key: C | c
    KEY_D               = 68,       // Key: D | d
    KEY_E               = 69,       // Key: E | e
    KEY_F               = 70,       // Key: F | f
    KEY_G               = 71,       // Key: G | g
    KEY_H               = 72,       // Key: H | h
    KEY_I               = 73,       // Key: I | i
    KEY_J               = 74,       // Key: J | j
    KEY_K               = 75,       // Key: K | k
    KEY_L               = 76,       // Key: L | l
    KEY_M               = 77,       // Key: M | m
    KEY_N               = 78,       // Key: N | n
    KEY_O               = 79,       // Key: O | o
    KEY_P               = 80,       // Key: P | p
    KEY_Q               = 81,       // Key: Q | q
    KEY_R               = 82,       // Key: R | r
    KEY_S               = 83,       // Key: S | s
    KEY_T               = 84,       // Key: T | t
    KEY_U               = 85,       // Key: U | u
    KEY_V               = 86,       // Key: V | v
    KEY_W               = 87,       // Key: W | w
    KEY_X               = 88,       // Key: X | x
    KEY_Y               = 89,       // Key: Y | y
    KEY_Z               = 90,       // Key: Z | z
    KEY_LEFT_BRACKET    = 91,       // Key: [
    KEY_BACKSLASH       = 92,       // Key: '\'
    KEY_RIGHT_BRACKET   = 93,       // Key: ]
    KEY_GRAVE           = 96,       // Key: `
    // Function keys
    KEY_SPACE           = 32,       // Key: Space
    KEY_ESCAPE          = 256,      // Key: Esc
    KEY_ENTER           = 257,      // Key: Enter
    KEY_TAB             = 258,      // Key: Tab
    KEY_BACKSPACE       = 259,      // Key: Backspace
    KEY_INSERT          = 260,      // Key: Ins
    KEY_DELETE          = 261,      // Key: Del
    KEY_RIGHT           = 262,      // Key: Cursor right
    KEY_LEFT            = 263,      // Key: Cursor left
    KEY_DOWN            = 264,      // Key: Cursor down
    KEY_UP              = 265,      // Key: Cursor up
    KEY_PAGE_UP         = 266,      // Key: Page up
    KEY_PAGE_DOWN       = 267,      // Key: Page down
    KEY_HOME            = 268,      // Key: Home
    KEY_END             = 269,      // Key: End
    KEY_CAPS_LOCK       = 280,      // Key: Caps lock
    KEY_SCROLL_LOCK     = 281,      // Key: Scroll down
    KEY_NUM_LOCK        = 282,      // Key: Num lock
    KEY_PRINT_SCREEN    = 283,      // Key: Print screen
    KEY_PAUSE           = 284,      // Key: Pause
    KEY_F1              = 290,      // Key: F1
    KEY_F2              = 291,      // Key: F2
    KEY_F3              = 292,      // Key: F3
    KEY_F4              = 293,      // Key: F4
    KEY_F5              = 294,      // Key: F5
    KEY_F6              = 295,      // Key: F6
    KEY_F7              = 296,      // Key: F7
    KEY_F8              = 297,      // Key: F8
    KEY_F9              = 298,      // Key: F9
    KEY_F10             = 299,      // Key: F10
    KEY_F11             = 300,      // Key: F11
    KEY_F12             = 301,      // Key: F12
    KEY_LEFT_SHIFT      = 340,      // Key: Shift left
    KEY_LEFT_CONTROL    = 341,      // Key: Control left
    KEY_LEFT_ALT        = 342,      // Key: Alt left
    KEY_LEFT_SUPER      = 343,      // Key: Super left
    KEY_RIGHT_SHIFT     = 344,      // Key: Shift right
    KEY_RIGHT_CONTROL   = 345,      // Key: Control right
    KEY_RIGHT_ALT       = 346,      // Key: Alt right
    KEY_RIGHT_SUPER     = 347,      // Key: Super right
    KEY_KB_MENU         = 348,      // Key: KB menu
    // Keypad keys
    KEY_KP_0            = 320,      // Key: Keypad 0
    KEY_KP_1            = 321,      // Key: Keypad 1
    KEY_KP_2            = 322,      // Key: Keypad 2
    KEY_KP_3            = 323,      // Key: Keypad 3
    KEY_KP_4            = 324,      // Key: Keypad 4
    KEY_KP_5            = 325,      // Key: Keypad 5
    KEY_KP_6            = 326,      // Key: Keypad 6
    KEY_KP_7            = 327,      // Key: Keypad 7
    KEY_KP_8            = 328,      // Key: Keypad 8
    KEY_KP_9            = 329,      // Key: Keypad 9
    KEY_KP_DECIMAL      = 330,      // Key: Keypad .
    KEY_KP_DIVIDE       = 331,      // Key: Keypad /
    KEY_KP_MULTIPLY     = 332,      // Key: Keypad *
    KEY_KP_SUBTRACT     = 333,      // Key: Keypad -
    KEY_KP_ADD          = 334,      // Key: Keypad +
    KEY_KP_ENTER        = 335,      // Key: Keypad Enter
    KEY_KP_EQUAL        = 336,      // Key: Keypad =
    // Android key buttons
    KEY_BACK            = 4,        // Key: Android back button
    KEY_MENU            = 5,        // Key: Android menu button
    KEY_VOLUME_UP       = 24,       // Key: Android volume up button
    KEY_VOLUME_DOWN     = 25        // Key: Android volume down button
} KeyboardKey;
```

enum MaterialMapIndex

Enumeradores:

MATERIAL_MAP_ALBEDO	
MATERIAL_MAP_METALNESS	
MATERIAL_MAP_NORMAL	
MATERIAL_MAP_ROUGHNESS	
MATERIAL_MAP_OCCLUSION	
MATERIAL_MAP_EMISSION	
MATERIAL_MAP_HEIGHT	
MATERIAL_MAP_CUBEMAP	

MATERIAL_MAP_IRRADIANCE	
MATERIAL_MAP_PREFILTER	
MATERIAL_MAP_BRDF	

```
             {
    MATERIAL_MAP_ALBEDO = 0,        // Albedo material (same as:
MATERIAL_MAP_DIFFUSE)
    MATERIAL_MAP_METALNESS,         // Metalness material (same as:
MATERIAL_MAP_SPECULAR)
    MATERIAL_MAP_NORMAL,            // Normal material
    MATERIAL_MAP_ROUGHNESS,         // Roughness material
    MATERIAL_MAP_OCCLUSION,         // Ambient occlusion material
    MATERIAL_MAP_EMISSION,          // Emission material
    MATERIAL_MAP_HEIGHT,            // Heightmap material
    MATERIAL_MAP_CUBEMAP,           // Cubemap material (NOTE: Uses
GL_TEXTURE_CUBE_MAP)
    MATERIAL_MAP_IRRADIANCE,        // Irradiance material (NOTE: Uses
GL_TEXTURE_CUBE_MAP)
    MATERIAL_MAP_PREFILTER,         // Prefilter material (NOTE: Uses
GL_TEXTURE_CUBE_MAP)
    MATERIAL_MAP_BRDF               // Brdf material
} MaterialMapIndex;
```

enum MouseButton

Enumeradores:

MOUSE_BUTTON_LEFT	
MOUSE_BUTTON_RIGHT	
MOUSE_BUTTON_MIDDLE	
MOUSE_BUTTON_SIDE	
MOUSE_BUTTON_EXTRA	
MOUSE_BUTTON_FORWARD	
MOUSE_BUTTON_BACK	

```
             {
    MOUSE_BUTTON_LEFT    = 0,       // Mouse button left
    MOUSE_BUTTON_RIGHT   = 1,       // Mouse button right
    MOUSE_BUTTON_MIDDLE  = 2,       // Mouse button middle (pressed wheel)
    MOUSE_BUTTON_SIDE    = 3,       // Mouse button side (advanced mouse device)
    MOUSE_BUTTON_EXTRA   = 4,       // Mouse button extra (advanced mouse device)
    MOUSE_BUTTON_FORWARD = 5,       // Mouse button forward (advanced mouse
device)
    MOUSE_BUTTON_BACK    = 6,       // Mouse button back (advanced mouse device)
} MouseButton;
```

enum MouseCursor

Enumeradores:

MOUSE_CURSOR_DEFAULT	
MOUSE_CURSOR_ARROW	
MOUSE_CURSOR_IBEAM	
MOUSE_CURSOR_CROSSHAIR	
MOUSE_CURSOR_POINTING_HAND	
MOUSE_CURSOR_RESIZE_EW	
MOUSE_CURSOR_RESIZE_NS	
MOUSE_CURSOR_RESIZE_NWSE	
MOUSE_CURSOR_RESIZE_NESW	
MOUSE_CURSOR_RESIZE_ALL	
MOUSE_CURSOR_NOT_ALLOWED	

```
            {
    MOUSE_CURSOR_DEFAULT       = 0,     // Default pointer shape
    MOUSE_CURSOR_ARROW         = 1,     // Arrow shape
    MOUSE_CURSOR_IBEAM         = 2,     // Text writing cursor shape
    MOUSE_CURSOR_CROSSHAIR     = 3,     // Cross shape
    MOUSE_CURSOR_POINTING_HAND = 4,     // Pointing hand cursor
    MOUSE_CURSOR_RESIZE_EW     = 5,     // Horizontal resize/move arrow shape
    MOUSE_CURSOR_RESIZE_NS     = 6,     // Vertical resize/move arrow shape
    MOUSE_CURSOR_RESIZE_NWSE   = 7,     // Top-left to bottom-right diagonal
resize/move arrow shape
    MOUSE_CURSOR_RESIZE_NESW   = 8,     // The top-right to bottom-left diagonal
resize/move arrow shape
    MOUSE_CURSOR_RESIZE_ALL    = 9,     // The omnidirectional resize/move cursor
shape
    MOUSE_CURSOR_NOT_ALLOWED   = 10     // The operation-not-allowed shape
} MouseCursor;
```

enum NPatchLayout

Enumeradores:

NPATCH_NINE_PATCH	
NPATCH_THREE_PATCH_VERTICAL	
NPATCH_THREE_PATCH_HORIZONTAL	

```
        {
    NPATCH_NINE_PATCH = 0,           // Npatch layout: 3x3 tiles
    NPATCH_THREE_PATCH_VERTICAL,     // Npatch layout: 1x3 tiles
    NPATCH_THREE_PATCH_HORIZONTAL    // Npatch layout: 3x1 tiles
} NPatchLayout;
```

enum PixelFormat

Enumeradores:

PIXELFORMAT_UNCOMPRESSED_GRAYSCALE	
PIXELFORMAT_UNCOMPRESSED_GRAY_ALPHA	
PIXELFORMAT_UNCOMPRESSED_R5G6B5	
PIXELFORMAT_UNCOMPRESSED_R8G8B8	
PIXELFORMAT_UNCOMPRESSED_R5G5B5A1	
PIXELFORMAT_UNCOMPRESSED_R4G4B4A4	
PIXELFORMAT_UNCOMPRESSED_R8G8B8A8	
PIXELFORMAT_UNCOMPRESSED_R32	
PIXELFORMAT_UNCOMPRESSED_R32G32B3	

2	
PIXELFORMAT_UNCOMPRESSED_R32G32B32A32	
PIXELFORMAT_UNCOMPRESSED_R16	
PIXELFORMAT_UNCOMPRESSED_R16G16B16	
PIXELFORMAT_UNCOMPRESSED_R16G16B16A16	
PIXELFORMAT_COMPRESSED_DXT1_RGB	
PIXELFORMAT_COMPRESSED_DXT1_RGBA	
PIXELFORMAT_COMPRESSED_DXT3_RGBA	
PIXELFORMAT_COMPRESSED_DXT5_RGBA	
PIXELFORMAT_COMPRESSED_ETC1_RGB	
PIXELFORMAT_COMPRESSED_ETC2_RGB	
PIXELFORMAT_COMPRESSED_ETC2_EAC_RGBA	
PIXELFORMAT_COMPRESSED_PVRT_RGB	
PIXELFORMAT_COMPRESSED_PVRT_RGBA	
PIXELFORMAT	

_COMPRESSED_ASTC_4x4_RGBA	
PIXELFORMAT_COMPRESSED_ASTC_8x8_RGBA	

```
        {
    PIXELFORMAT_UNCOMPRESSED_GRAYSCALE = 1, // 8 bit per pixel (no alpha)
    PIXELFORMAT_UNCOMPRESSED_GRAY_ALPHA,    // 8*2 bpp (2 channels)
    PIXELFORMAT_UNCOMPRESSED_R5G6B5,        // 16 bpp
    PIXELFORMAT_UNCOMPRESSED_R8G8B8,        // 24 bpp
    PIXELFORMAT_UNCOMPRESSED_R5G5B5A1,      // 16 bpp (1 bit alpha)
    PIXELFORMAT_UNCOMPRESSED_R4G4B4A4,      // 16 bpp (4 bit alpha)
    PIXELFORMAT_UNCOMPRESSED_R8G8B8A8,      // 32 bpp
    PIXELFORMAT_UNCOMPRESSED_R32,           // 32 bpp (1 channel - float)
    PIXELFORMAT_UNCOMPRESSED_R32G32B32,     // 32*3 bpp (3 channels - float)
    PIXELFORMAT_UNCOMPRESSED_R32G32B32A32,  // 32*4 bpp (4 channels - float)
    PIXELFORMAT_UNCOMPRESSED_R16,           // 16 bpp (1 channel - half float)
    PIXELFORMAT_UNCOMPRESSED_R16G16B16,     // 16*3 bpp (3 channels - half float)
    PIXELFORMAT_UNCOMPRESSED_R16G16B16A16,  // 16*4 bpp (4 channels - half float)
    PIXELFORMAT_COMPRESSED_DXT1_RGB,        // 4 bpp (no alpha)
    PIXELFORMAT_COMPRESSED_DXT1_RGBA,       // 4 bpp (1 bit alpha)
    PIXELFORMAT_COMPRESSED_DXT3_RGBA,       // 8 bpp
    PIXELFORMAT_COMPRESSED_DXT5_RGBA,       // 8 bpp
    PIXELFORMAT_COMPRESSED_ETC1_RGB,        // 4 bpp
    PIXELFORMAT_COMPRESSED_ETC2_RGB,        // 4 bpp
    PIXELFORMAT_COMPRESSED_ETC2_EAC_RGBA,   // 8 bpp
    PIXELFORMAT_COMPRESSED_PVRT_RGB,        // 4 bpp
    PIXELFORMAT_COMPRESSED_PVRT_RGBA,       // 4 bpp
    PIXELFORMAT_COMPRESSED_ASTC_4x4_RGBA,   // 8 bpp
    PIXELFORMAT_COMPRESSED_ASTC_8x8_RGBA    // 2 bpp
} PixelFormat;
```

enum ShaderAttributeDataType

Enumeradores:

SHADER_ATTRIB_FLOAT	
SHADER_ATTRIB_VEC2	
SHADER_ATTRIB_VEC3	
SHADER_ATTRIB_VEC4	

```
        {
    SHADER_ATTRIB_FLOAT = 0,        // Shader attribute type: float
    SHADER_ATTRIB_VEC2,             // Shader attribute type: vec2 (2 float)
    SHADER_ATTRIB_VEC3,             // Shader attribute type: vec3 (3 float)
    SHADER_ATTRIB_VEC4              // Shader attribute type: vec4 (4 float)
} ShaderAttributeDataType;
```

enum ShaderLocationIndex

Enumeradores:

SHADER_LOC_VERTEX_POSI	

TION	
SHADER_LOC_ VERTEX_TEX COORD01	
SHADER_LOC_ VERTEX_TEX COORD02	
SHADER_LOC_ VERTEX_NOR MAL	
SHADER_LOC_ VERTEX_TAN GENT	
SHADER_LOC_ VERTEX_COL OR	
SHADER_LOC_ MATRIX_MVP	
SHADER_LOC_ MATRIX_VIE W	
SHADER_LOC_ MATRIX_PROJ ECTION	
SHADER_LOC_ MATRIX_MOD EL	
SHADER_LOC_ MATRIX_NOR MAL	
SHADER_LOC_ VECTOR_VIE W	
SHADER_LOC_ COLOR_DIFFU SE	
SHADER_LOC_ COLOR_SPECU LAR	
SHADER_LOC_ COLOR_AMBI ENT	
SHADER_LOC_ MAP_ALBEDO	
SHADER_LOC_ MAP_METALN ESS	
SHADER_LOC_	

MAP_NORMAL	
SHADER_LOC_MAP_ROUGHNESS	
SHADER_LOC_MAP_OCCLUSION	
SHADER_LOC_MAP_EMISSION	
SHADER_LOC_MAP_HEIGHT	
SHADER_LOC_MAP_CUBEMAP	
SHADER_LOC_MAP_IRRADIANCE	
SHADER_LOC_MAP_PREFILTER	
SHADER_LOC_MAP_BRDF	
SHADER_LOC_VERTEX_BONEIDS	
SHADER_LOC_VERTEX_BONEWEIGHTS	
SHADER_LOC_BONE_MATRICES	

```
                {
        SHADER_LOC_VERTEX_POSITION = 0, // Shader location: vertex attribute: position
        SHADER_LOC_VERTEX_TEXCOORD01,   // Shader location: vertex attribute: texcoord01
        SHADER_LOC_VERTEX_TEXCOORD02,   // Shader location: vertex attribute: texcoord02
        SHADER_LOC_VERTEX_NORMAL,       // Shader location: vertex attribute: normal
        SHADER_LOC_VERTEX_TANGENT,      // Shader location: vertex attribute: tangent
        SHADER_LOC_VERTEX_COLOR,        // Shader location: vertex attribute: color
        SHADER_LOC_MATRIX_MVP,          // Shader location: matrix uniform: model-view-projection
        SHADER_LOC_MATRIX_VIEW,         // Shader location: matrix uniform: view (camera transform)
        SHADER_LOC_MATRIX_PROJECTION,   // Shader location: matrix uniform: projection
        SHADER_LOC_MATRIX_MODEL,        // Shader location: matrix uniform: model (transform)
        SHADER_LOC_MATRIX_NORMAL,       // Shader location: matrix uniform: normal
        SHADER_LOC_VECTOR_VIEW,         // Shader location: vector uniform: view
        SHADER_LOC_COLOR_DIFFUSE,       // Shader location: vector uniform: diffuse color
        SHADER_LOC_COLOR_SPECULAR,      // Shader location: vector uniform: specular color
```

```
    SHADER_LOC_COLOR_AMBIENT,        // Shader location: vector uniform: ambient color
    SHADER_LOC_MAP_ALBEDO,           // Shader location: sampler2d texture: albedo (same as: SHADER_LOC_MAP_DIFFUSE)
    SHADER_LOC_MAP_METALNESS,        // Shader location: sampler2d texture: metalness (same as: SHADER_LOC_MAP_SPECULAR)
    SHADER_LOC_MAP_NORMAL,           // Shader location: sampler2d texture: normal
    SHADER_LOC_MAP_ROUGHNESS,        // Shader location: sampler2d texture: roughness
    SHADER_LOC_MAP_OCCLUSION,        // Shader location: sampler2d texture: occlusion
    SHADER_LOC_MAP_EMISSION,         // Shader location: sampler2d texture: emission
    SHADER_LOC_MAP_HEIGHT,           // Shader location: sampler2d texture: height
    SHADER_LOC_MAP_CUBEMAP,          // Shader location: samplerCube texture: cubemap
    SHADER_LOC_MAP_IRRADIANCE,       // Shader location: samplerCube texture: irradiance
    SHADER_LOC_MAP_PREFILTER,        // Shader location: samplerCube texture: prefilter
    SHADER_LOC_MAP_BRDF,             // Shader location: sampler2d texture: brdf
    SHADER_LOC_VERTEX_BONEIDS,       // Shader location: vertex attribute: boneIds
    SHADER_LOC_VERTEX_BONEWEIGHTS,   // Shader location: vertex attribute: boneWeights
    SHADER_LOC_BONE_MATRICES         // Shader location: array of matrices uniform: boneMatrices
} ShaderLocationIndex;
```

enum ShaderUniformDataType

Enumeradores:

SHADER_UNIFORM_FLOAT	
SHADER_UNIFORM_VEC2	
SHADER_UNIFORM_VEC3	
SHADER_UNIFORM_VEC4	
SHADER_UNIFORM_INT	
SHADER_UNIFORM_IVEC2	
SHADER_UNIFORM_IVEC3	
SHADER_UNIFORM_IVEC4	
SHADER_UNIFORM_SAMPLER2D	

```
            {
    SHADER_UNIFORM_FLOAT = 0,        // Shader uniform type: float
    SHADER_UNIFORM_VEC2,             // Shader uniform type: vec2 (2 float)
    SHADER_UNIFORM_VEC3,             // Shader uniform type: vec3 (3 float)
    SHADER_UNIFORM_VEC4,             // Shader uniform type: vec4 (4 float)
    SHADER_UNIFORM_INT,              // Shader uniform type: int
    SHADER_UNIFORM_IVEC2,            // Shader uniform type: ivec2 (2 int)
    SHADER_UNIFORM_IVEC3,            // Shader uniform type: ivec3 (3 int)
    SHADER_UNIFORM_IVEC4,            // Shader uniform type: ivec4 (4 int)
    SHADER_UNIFORM_SAMPLER2D         // Shader uniform type: sampler2d
```

```
} ShaderUniformDataType;
```

enum TextureFilter

Enumeradores:

TEXTURE_FILTER_POINT	
TEXTURE_FILTER_BILINEAR	
TEXTURE_FILTER_TRILINEAR	
TEXTURE_FILTER_ANISOTROPIC_4X	
TEXTURE_FILTER_ANISOTROPIC_8X	
TEXTURE_FILTER_ANISOTROPIC_16X	

```
                {
    TEXTURE_FILTER_POINT = 0,                  // No filter, just pixel approximation
    TEXTURE_FILTER_BILINEAR,                    // Linear filtering
    TEXTURE_FILTER_TRILINEAR,                   // Trilinear filtering (linear with
mipmaps)
    TEXTURE_FILTER_ANISOTROPIC_4X,              // Anisotropic filtering 4x
    TEXTURE_FILTER_ANISOTROPIC_8X,              // Anisotropic filtering 8x
    TEXTURE_FILTER_ANISOTROPIC_16X,             // Anisotropic filtering 16x
} TextureFilter;
```

enum TextureWrap

Enumeradores:

TEXTURE_WRAP_REPEAT	
TEXTURE_WRAP_CLAMP	
TEXTURE_WRAP_MIRROR_REPEAT	
TEXTURE_WRAP_MIRROR_CLAMP	

```
                {
    TEXTURE_WRAP_REPEAT = 0,                    // Repeats texture in tiled mode
    TEXTURE_WRAP_CLAMP,                         // Clamps texture to edge pixel in
tiled mode
    TEXTURE_WRAP_MIRROR_REPEAT,                 // Mirrors and repeats the texture in
tiled mode
```

```
    TEXTURE_WRAP_MIRROR_CLAMP                // Mirrors and clamps to border the
texture in tiled mode
} TextureWrap;
```

enum TraceLogLevel

Enumeradores:

LOG_ALL	
LOG_TRACE	
LOG_DEBUG	
LOG_INFO	
LOG_WARNING	
LOG_ERROR	
LOG_FATAL	
LOG_NONE	

```
            {
    LOG_ALL = 0,          // Display all logs
    LOG_TRACE,            // Trace logging, intended for internal use only
    LOG_DEBUG,            // Debug logging, used for internal debugging, it should
be disabled on release builds
    LOG_INFO,             // Info logging, used for program execution info
    LOG_WARNING,          // Warning logging, used on recoverable failures
    LOG_ERROR,            // Error logging, used on unrecoverable failures
    LOG_FATAL,            // Fatal logging, used to abort program:
exit(EXIT_FAILURE)
    LOG_NONE              // Disable logging
} TraceLogLevel;
```

Funções

RLAPI void AttachAudioMixedProcessor (AudioCallback processor)

RLAPI void AttachAudioStreamProcessor (AudioStream stream, AudioCallback processor)

RLAPI void BeginBlendMode (int mode)

RLAPI void BeginDrawing (void)

RLAPI void BeginMode2D (Camera2D camera)

RLAPI void BeginMode3D (Camera3D camera)

RLAPI void BeginScissorMode (int x, int y, int width, int height)

RLAPI void BeginShaderMode (Shader shader)

RLAPI void BeginTextureMode (RenderTexture2D target)

RLAPI void BeginVrStereoMode (VrStereoConfig config)

RLAPI bool ChangeDirectory (const char * dir)

RLAPI bool CheckCollisionBoxes (BoundingBox box1, BoundingBox box2)

RLAPI bool CheckCollisionBoxSphere (BoundingBox box, Vector3 center, float radius)

RLAPI bool CheckCollisionCircleLine (Vector2 center, float radius, Vector2 p1, Vector2 p2)

RLAPI bool CheckCollisionCircleRec (Vector2 center, float radius, Rectangle rec)

RLAPI bool CheckCollisionCircles (Vector2 center1, float radius1, Vector2 center2, float radius2)

RLAPI bool CheckCollisionLines (Vector2 startPos1, Vector2 endPos1, Vector2 startPos2, Vector2 endPos2, Vector2 * collisionPoint)

RLAPI bool CheckCollisionPointCircle (Vector2 point, Vector2 center, float radius)

RLAPI bool CheckCollisionPointLine (Vector2 point, Vector2 p1, Vector2 p2, int threshold)

RLAPI bool CheckCollisionPointPoly (Vector2 point, const Vector2 * points, int pointCount)

RLAPI bool CheckCollisionPointRec (Vector2 point, Rectangle rec)

RLAPI bool CheckCollisionPointTriangle (Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3)

RLAPI bool CheckCollisionRecs (Rectangle rec1, Rectangle rec2)

RLAPI bool CheckCollisionSpheres (Vector3 center1, float radius1, Vector3 center2, float radius2)

RLAPI void ClearBackground (Color color)

RLAPI void ClearWindowState (unsigned int flags)

RLAPI void CloseAudioDevice (void)

RLAPI void CloseWindow (void)

RLAPI const char * CodepointToUTF8 (int codepoint, int * utf8Size)

RLAPI Color ColorAlpha (Color color, float alpha)

RLAPI Color ColorAlphaBlend (Color dst, Color src, Color tint)

RLAPI Color ColorBrightness (Color color, float factor)

RLAPI Color ColorContrast (Color color, float contrast)

RLAPI Color ColorFromHSV (float hue, float saturation, float value)

RLAPI Color ColorFromNormalized (Vector4 normalized)

RLAPI bool ColorIsEqual (Color col1, Color col2)

RLAPI Color ColorLerp (Color color1, Color color2, float factor)

RLAPI Vector4 ColorNormalize (Color color)

RLAPI Color ColorTint (Color color, Color tint)

RLAPI Vector3 ColorToHSV (Color color)

RLAPI int ColorToInt (Color color)

RLAPI unsigned char * CompressData (const unsigned char * data, int dataSize, int * compDataSize)

RLAPI unsigned int ComputeCRC32 (unsigned char * data, int dataSize)

RLAPI unsigned int * ComputeMD5 (unsigned char * data, int dataSize)

RLAPI unsigned int * ComputeSHA1 (unsigned char * data, int dataSize)

RLAPI unsigned char * DecodeDataBase64 (const unsigned char * data, int * outputSize)

RLAPI unsigned char * DecompressData (const unsigned char * compData, int compDataSize, int * dataSize)

RLAPI void DetachAudioMixedProcessor (AudioCallback processor)

RLAPI void DetachAudioStreamProcessor (AudioStream stream, AudioCallback processor)

RLAPI bool DirectoryExists (const char * dirPath)

RLAPI void DisableCursor (void)

RLAPI void DisableEventWaiting (void)

RLAPI void DrawBillboard (Camera camera, Texture2D texture, Vector3 position, float scale, Color tint)

RLAPI void DrawBillboardPro (Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector3 up, Vector2 size, Vector2 origin, float rotation, Color tint)

RLAPI void DrawBillboardRec (Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector2 size, Color tint)

RLAPI void DrawBoundingBox (BoundingBox box, Color color)

RLAPI void DrawCapsule (Vector3 startPos, Vector3 endPos, float radius, int slices, int rings, Color color)

RLAPI void DrawCapsuleWires (Vector3 startPos, Vector3 endPos, float radius, int slices, int rings, Color color)

RLAPI void DrawCircle (int centerX, int centerY, float radius, Color color)

RLAPI void DrawCircle3D (Vector3 center, float radius, Vector3 rotationAxis, float rotationAngle, Color color)

RLAPI void DrawCircleGradient (int centerX, int centerY, float radius, Color inner, Color outer)

RLAPI void DrawCircleLines (int centerX, int centerY, float radius, Color color)

RLAPI void DrawCircleLinesV (Vector2 center, float radius, Color color)

RLAPI void DrawCircleSector (Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color)

RLAPI void DrawCircleSectorLines (Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color)

RLAPI void DrawCircleV (Vector2 center, float radius, Color color)

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

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

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

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

RLAPI void DrawCylinder (Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color)

RLAPI void DrawCylinderEx (Vector3 startPos, Vector3 endPos, float startRadius, float endRadius, int sides, Color color)

RLAPI void DrawCylinderWires (Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color)

RLAPI void DrawCylinderWiresEx (Vector3 startPos, Vector3 endPos, float startRadius, float endRadius, int sides, Color color)

RLAPI void DrawEllipse (int centerX, int centerY, float radiusH, float radiusV, Color color)

RLAPI void DrawEllipseLines (int centerX, int centerY, float radiusH, float radiusV, Color color)

RLAPI void DrawFPS (int posX, int posY)

RLAPI void DrawGrid (int slices, float spacing)

RLAPI void DrawLine (int startPosX, int startPosY, int endPosX, int endPosY, Color color)

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

RLAPI void DrawLineBezier (Vector2 startPos, Vector2 endPos, float thick, Color color)

RLAPI void DrawLineEx (Vector2 startPos, Vector2 endPos, float thick, Color color)

RLAPI void DrawLineStrip (const Vector2 * points, int pointCount, Color color)

RLAPI void DrawLineV (Vector2 startPos, Vector2 endPos, Color color)

RLAPI void DrawMesh (Mesh mesh, Material material, Matrix transform)

RLAPI void DrawMeshInstanced (Mesh mesh, Material material, const Matrix * transforms, int instances)

RLAPI void DrawModel (Model model, Vector3 position, float scale, Color tint)

RLAPI void DrawModelEx (Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint)

RLAPI void DrawModelPoints (Model model, Vector3 position, float scale, Color tint)

RLAPI void DrawModelPointsEx (Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint)

RLAPI void DrawModelWires (Model model, Vector3 position, float scale, Color tint)

RLAPI void DrawModelWiresEx (Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint)

RLAPI void DrawPixel (int posX, int posY, Color color)

RLAPI void DrawPixelV (Vector2 position, Color color)

RLAPI void DrawPlane (Vector3 centerPos, Vector2 size, Color color)

RLAPI void DrawPoint3D (Vector3 position, Color color)

RLAPI void DrawPoly (Vector2 center, int sides, float radius, float rotation, Color color)

RLAPI void DrawPolyLines (Vector2 center, int sides, float radius, float rotation, Color color)

RLAPI void DrawPolyLinesEx (Vector2 center, int sides, float radius, float rotation, float lineThick, Color color)

RLAPI void DrawRay (Ray ray, Color color)

RLAPI void DrawRectangle (int posX, int posY, int width, int height, Color color)

RLAPI void DrawRectangleGradientEx (Rectangle rec, Color topLeft, Color bottomLeft, Color topRight, Color bottomRight)

RLAPI void DrawRectangleGradientH (int posX, int posY, int width, int height, Color left, Color right)

RLAPI void DrawRectangleGradientV (int posX, int posY, int width, int height, Color top, Color bottom)

RLAPI void DrawRectangleLines (int posX, int posY, int width, int height, Color color)

RLAPI void DrawRectangleLinesEx (Rectangle rec, float lineThick, Color color)

RLAPI void DrawRectanglePro (Rectangle rec, Vector2 origin, float rotation, Color color)

RLAPI void DrawRectangleRec (Rectangle rec, Color color)

RLAPI void DrawRectangleRounded (Rectangle rec, float roundness, int segments, Color color)

RLAPI void DrawRectangleRoundedLines (Rectangle rec, float roundness, int segments, Color color)

RLAPI void DrawRectangleRoundedLinesEx (Rectangle rec, float roundness, int segments, float lineThick, Color color)

RLAPI void DrawRectangleV (Vector2 position, Vector2 size, Color color)

RLAPI void DrawRing (Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color)

RLAPI void DrawRingLines (Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color)

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

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

RLAPI void DrawSphereWires (Vector3 centerPos, float radius, int rings, int slices, Color color)

RLAPI void DrawSplineBasis (const Vector2 * points, int pointCount, float thick, Color color)

RLAPI void DrawSplineBezierCubic (const Vector2 * points, int pointCount, float thick, Color color)

RLAPI void DrawSplineBezierQuadratic (const Vector2 * points, int pointCount, float thick, Color color)

RLAPI void DrawSplineCatmullRom (const Vector2 * points, int pointCount, float thick, Color color)

RLAPI void DrawSplineLinear (const Vector2 * points, int pointCount, float thick, Color color)

RLAPI void DrawSplineSegmentBasis (Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color)

RLAPI void DrawSplineSegmentBezierCubic (Vector2 p1, Vector2 c2, Vector2 c3, Vector2 p4, float thick, Color color)

RLAPI void DrawSplineSegmentBezierQuadratic (Vector2 p1, Vector2 c2, Vector2 p3, float thick, Color color)

RLAPI void DrawSplineSegmentCatmullRom (Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color)

RLAPI void DrawSplineSegmentLinear (Vector2 p1, Vector2 p2, float thick, Color color)

RLAPI void DrawText (const char * text, int posX, int posY, int fontSize, Color color)

RLAPI void DrawTextCodepoint (Font font, int codepoint, Vector2 position, float fontSize, Color tint)

RLAPI void DrawTextCodepoints (Font font, const int * codepoints, int codepointCount, Vector2 position, float fontSize, float spacing, Color tint)

RLAPI void DrawTextEx (Font font, const char * text, Vector2 position, float fontSize, float spacing, Color tint)

RLAPI void DrawTextPro (Font font, const char * text, Vector2 position, Vector2 origin, float rotation, float fontSize, float spacing, Color tint)

RLAPI void DrawTexture (Texture2D texture, int posX, int posY, Color tint)

RLAPI void DrawTextureEx (Texture2D texture, Vector2 position, float rotation, float scale, Color tint)

RLAPI void DrawTextureNPatch (Texture2D texture, NPatchInfo nPatchInfo, Rectangle dest, Vector2 origin, float rotation, Color tint)

RLAPI void DrawTexturePro (Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, Color tint)

RLAPI void DrawTextureRec (Texture2D texture, Rectangle source, Vector2 position, Color tint)

RLAPI void DrawTextureV (Texture2D texture, Vector2 position, Color tint)

RLAPI void DrawTriangle (Vector2 v1, Vector2 v2, Vector2 v3, Color color)

RLAPI void DrawTriangle3D (Vector3 v1, Vector3 v2, Vector3 v3, Color color)

RLAPI void DrawTriangleFan (const Vector2 * points, int pointCount, Color color)

RLAPI void DrawTriangleLines (Vector2 v1, Vector2 v2, Vector2 v3, Color color)

RLAPI void DrawTriangleStrip (const Vector2 * points, int pointCount, Color color)

RLAPI void DrawTriangleStrip3D (const Vector3 * points, int pointCount, Color color)

RLAPI void EnableCursor (void)

RLAPI void EnableEventWaiting (void)

RLAPI char * EncodeDataBase64 (const unsigned char * data, int dataSize, int * outputSize)

RLAPI void EndBlendMode (void)

RLAPI void EndDrawing (void)

RLAPI void EndMode2D (void)

RLAPI void EndMode3D (void)

RLAPI void EndScissorMode (void)

RLAPI void EndShaderMode (void)

RLAPI void EndTextureMode (void)

RLAPI void EndVrStereoMode (void)

RLAPI bool ExportAutomationEventList (AutomationEventList list, const char * fileName)

RLAPI bool ExportDataAsCode (const unsigned char * data, int dataSize, const char * fileName)

RLAPI bool ExportFontAsCode (Font font, const char * fileName)

RLAPI bool ExportImage (Image image, const char * fileName)

RLAPI bool ExportImageAsCode (Image image, const char * fileName)

RLAPI unsigned char * ExportImageToMemory (Image image, const char * fileType, int * fileSize)

RLAPI bool ExportMesh (Mesh mesh, const char * fileName)

RLAPI bool ExportMeshAsCode (Mesh mesh, const char * fileName)

RLAPI bool ExportWave (Wave wave, const char * fileName)

RLAPI bool ExportWaveAsCode (Wave wave, const char * fileName)

RLAPI Color Fade (Color color, float alpha)

RLAPI bool FileExists (const char * fileName)

RLAPI Image GenImageCellular (int width, int height, int tileSize)

RLAPI Image GenImageChecked (int width, int height, int checksX, int checksY, Color col1, Color col2)

RLAPI Image GenImageColor (int width, int height, Color color)

RLAPI Image GenImageFontAtlas (const GlyphInfo * glyphs, Rectangle ** glyphRecs, int glyphCount, int fontSize, int padding, int packMethod)

RLAPI Image GenImageGradientLinear (int width, int height, int direction, Color start, Color end)

RLAPI Image GenImageGradientRadial (int width, int height, float density, Color inner, Color outer)

RLAPI Image GenImageGradientSquare (int width, int height, float density, Color inner, Color outer)

RLAPI Image GenImagePerlinNoise (int width, int height, int offsetX, int offsetY, float scale)

RLAPI Image GenImageText (int width, int height, const char * text)

RLAPI Image GenImageWhiteNoise (int width, int height, float factor)

RLAPI Mesh GenMeshCone (float radius, float height, int slices)

RLAPI Mesh GenMeshCube (float width, float height, float length)

RLAPI Mesh GenMeshCubicmap (Image cubicmap, Vector3 cubeSize)

RLAPI Mesh GenMeshCylinder (float radius, float height, int slices)

RLAPI Mesh GenMeshHeightmap (Image heightmap, Vector3 size)

RLAPI Mesh GenMeshHemiSphere (float radius, int rings, int slices)

RLAPI Mesh GenMeshKnot (float radius, float size, int radSeg, int sides)

RLAPI Mesh GenMeshPlane (float width, float length, int resX, int resZ)

RLAPI Mesh GenMeshPoly (int sides, float radius)

RLAPI Mesh GenMeshSphere (float radius, int rings, int slices)

RLAPI void GenMeshTangents (Mesh * mesh)

RLAPI Mesh GenMeshTorus (float radius, float size, int radSeg, int sides)

RLAPI void GenTextureMipmaps (Texture2D * texture)

RLAPI const char * GetApplicationDirectory (void)

RLAPI Matrix GetCameraMatrix (Camera camera)

RLAPI Matrix GetCameraMatrix2D (Camera2D camera)

RLAPI int GetCharPressed (void)

RLAPI Image GetClipboardImage (void)

RLAPI const char * GetClipboardText (void)

RLAPI int GetCodepoint (const char * text, int * codepointSize)

RLAPI int GetCodepointCount (const char * text)

RLAPI int GetCodepointNext (const char * text, int * codepointSize)

RLAPI int GetCodepointPrevious (const char * text, int * codepointSize)

RLAPI Rectangle GetCollisionRec (Rectangle rec1, Rectangle rec2)

RLAPI Color GetColor (unsigned int hexValue)

RLAPI int GetCurrentMonitor (void)

RLAPI const char * GetDirectoryPath (const char * filePath)

RLAPI const char * GetFileExtension (const char * fileName)

RLAPI int GetFileLength (const char * fileName)

RLAPI long GetFileModTime (const char * fileName)

RLAPI const char * GetFileName (const char * filePath)

RLAPI const char * GetFileNameWithoutExt (const char * filePath)

RLAPI Font GetFontDefault (void)

RLAPI int GetFPS (void)

RLAPI float GetFrameTime (void)

RLAPI int GetGamepadAxisCount (int gamepad)

RLAPI float GetGamepadAxisMovement (int gamepad, int axis)

RLAPI int GetGamepadButtonPressed (void)

RLAPI const char * GetGamepadName (int gamepad)

RLAPI int GetGestureDetected (void)

RLAPI float GetGestureDragAngle (void)

RLAPI Vector2 GetGestureDragVector (void)

RLAPI float GetGestureHoldDuration (void)

RLAPI float GetGesturePinchAngle (void)

RLAPI Vector2 GetGesturePinchVector (void)

RLAPI Rectangle GetGlyphAtlasRec (Font font, int codepoint)

RLAPI int GetGlyphIndex (Font font, int codepoint)

RLAPI GlyphInfo GetGlyphInfo (Font font, int codepoint)

RLAPI Rectangle GetImageAlphaBorder (Image image, float threshold)

RLAPI Color GetImageColor (Image image, int x, int y)

RLAPI int GetKeyPressed (void)

RLAPI float GetMasterVolume (void)

RLAPI BoundingBox GetMeshBoundingBox (Mesh mesh)

RLAPI BoundingBox GetModelBoundingBox (Model model)

RLAPI int GetMonitorCount (void)

RLAPI int GetMonitorHeight (int monitor)

RLAPI const char * GetMonitorName (int monitor)

RLAPI int GetMonitorPhysicalHeight (int monitor)

RLAPI int GetMonitorPhysicalWidth (int monitor)

RLAPI Vector2 GetMonitorPosition (int monitor)

RLAPI int GetMonitorRefreshRate (int monitor)

RLAPI int GetMonitorWidth (int monitor)

RLAPI Vector2 GetMouseDelta (void)

RLAPI Vector2 GetMousePosition (void)

RLAPI float GetMouseWheelMove (void)

RLAPI Vector2 GetMouseWheelMoveV (void)

RLAPI int GetMouseX (void)

RLAPI int GetMouseY (void)

RLAPI float GetMusicTimeLength (Music music)

RLAPI float GetMusicTimePlayed (Music music)

RLAPI Color GetPixelColor (void * srcPtr, int format)

RLAPI int GetPixelDataSize (int width, int height, int format)

RLAPI const char * GetPrevDirectoryPath (const char * dirPath)

RLAPI int GetRandomValue (int min, int max)

RLAPI RayCollision GetRayCollisionBox (Ray ray, BoundingBox box)

RLAPI RayCollision GetRayCollisionMesh (Ray ray, Mesh mesh, Matrix transform)

RLAPI RayCollision GetRayCollisionQuad (Ray ray, Vector3 p1, Vector3 p2, Vector3 p3, Vector3 p4)

RLAPI RayCollision GetRayCollisionSphere (Ray ray, Vector3 center, float radius)

RLAPI RayCollision GetRayCollisionTriangle (Ray ray, Vector3 p1, Vector3 p2, Vector3 p3)

RLAPI int GetRenderHeight (void)

RLAPI int GetRenderWidth (void)

RLAPI int GetScreenHeight (void)

RLAPI Vector2 GetScreenToWorld2D (Vector2 position, Camera2D camera)

RLAPI Ray GetScreenToWorldRay (Vector2 position, Camera camera)

RLAPI Ray GetScreenToWorldRayEx (Vector2 position, Camera camera, int width, int height)

RLAPI int GetScreenWidth (void)

RLAPI int GetShaderLocation (Shader shader, const char * uniformName)

RLAPI int GetShaderLocationAttrib (Shader shader, const char * attribName)

RLAPI Texture2D GetShapesTexture (void)

RLAPI Rectangle GetShapesTextureRectangle (void)

RLAPI Vector2 GetSplinePointBasis (Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float t)

RLAPI Vector2 GetSplinePointBezierCubic (Vector2 p1, Vector2 c2, Vector2 c3, Vector2 p4, float t)

RLAPI Vector2 GetSplinePointBezierQuad (Vector2 p1, Vector2 c2, Vector2 p3, float t)

RLAPI Vector2 GetSplinePointCatmullRom (Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float t)

RLAPI Vector2 GetSplinePointLinear (Vector2 startPos, Vector2 endPos, float t)

RLAPI double GetTime (void)

RLAPI int GetTouchPointCount (void)

RLAPI int GetTouchPointId (int index)

RLAPI Vector2 GetTouchPosition (int index)

RLAPI int GetTouchX (void)

RLAPI int GetTouchY (void)

RLAPI void * GetWindowHandle (void)

RLAPI Vector2 GetWindowPosition (void)

RLAPI Vector2 GetWindowScaleDPI (void)

RLAPI const char * GetWorkingDirectory (void)

RLAPI Vector2 GetWorldToScreen (Vector3 position, Camera camera)

RLAPI Vector2 GetWorldToScreen2D (Vector2 position, Camera2D camera)

RLAPI Vector2 GetWorldToScreenEx (Vector3 position, Camera camera, int width, int height)

RLAPI void HideCursor (void)

RLAPI void ImageAlphaClear (Image * image, Color color, float threshold)

RLAPI void ImageAlphaCrop (Image * image, float threshold)

RLAPI void ImageAlphaMask (Image * image, Image alphaMask)

RLAPI void ImageAlphaPremultiply (Image * image)

RLAPI void ImageBlurGaussian (Image * image, int blurSize)

RLAPI void ImageClearBackground (Image * dst, Color color)

RLAPI void ImageColorBrightness (Image * image, int brightness)

RLAPI void ImageColorContrast (Image * image, float contrast)

RLAPI void ImageColorGrayscale (Image * image)

RLAPI void ImageColorInvert (Image * image)

RLAPI void ImageColorReplace (Image * image, Color color, Color replace)

RLAPI void ImageColorTint (Image * image, Color color)

RLAPI Image ImageCopy (Image image)

RLAPI void ImageCrop (Image * image, Rectangle crop)

RLAPI void ImageDither (Image * image, int rBpp, int gBpp, int bBpp, int aBpp)

RLAPI void ImageDraw (Image * dst, Image src, Rectangle srcRec, Rectangle dstRec, Color tint)

RLAPI void ImageDrawCircle (Image * dst, int centerX, int centerY, int radius, Color color)

RLAPI void ImageDrawCircleLines (Image * dst, int centerX, int centerY, int radius, Color color)

RLAPI void ImageDrawCircleLinesV (Image * dst, Vector2 center, int radius, Color color)

RLAPI void ImageDrawCircleV (Image * dst, Vector2 center, int radius, Color color)

RLAPI void ImageDrawLine (Image * dst, int startPosX, int startPosY, int endPosX, int endPosY, Color color)

RLAPI void ImageDrawLineEx (Image * dst, Vector2 start, Vector2 end, int thick, Color color)

RLAPI void ImageDrawLineV (Image * dst, Vector2 start, Vector2 end, Color color)

RLAPI void ImageDrawPixel (Image * dst, int posX, int posY, Color color)

RLAPI void ImageDrawPixelV (Image * dst, Vector2 position, Color color)

RLAPI void ImageDrawRectangle (Image * dst, int posX, int posY, int width, int height, Color color)

RLAPI void ImageDrawRectangleLines (Image * dst, Rectangle rec, int thick, Color color)

RLAPI void ImageDrawRectangleRec (Image * dst, Rectangle rec, Color color)

RLAPI void ImageDrawRectangleV (Image * dst, Vector2 position, Vector2 size, Color color)

RLAPI void ImageDrawText (Image * dst, const char * text, int posX, int posY, int fontSize, Color color)

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

RLAPI void ImageDrawTriangle (Image * dst, Vector2 v1, Vector2 v2, Vector2 v3, Color color)

RLAPI void ImageDrawTriangleEx (Image * dst, Vector2 v1, Vector2 v2, Vector2 v3, Color c1, Color c2, Color c3)

RLAPI void ImageDrawTriangleFan (Image * dst, Vector2 * points, int pointCount, Color color)

RLAPI void ImageDrawTriangleLines (Image * dst, Vector2 v1, Vector2 v2, Vector2 v3, Color color)

RLAPI void ImageDrawTriangleStrip (Image * dst, Vector2 * points, int pointCount, Color color)

RLAPI void ImageFlipHorizontal (Image * image)

RLAPI void ImageFlipVertical (Image * image)

RLAPI void ImageFormat (Image * image, int newFormat)

RLAPI Image ImageFromChannel (Image image, int selectedChannel)

RLAPI Image ImageFromImage (Image image, Rectangle rec)

RLAPI void ImageKernelConvolution (Image * image, const float * kernel, int kernelSize)

RLAPI void ImageMipmaps (Image * image)

RLAPI void ImageResize (Image * image, int newWidth, int newHeight)

RLAPI void ImageResizeCanvas (Image * image, int newWidth, int newHeight, int offsetX, int offsetY, Color fill)

RLAPI void ImageResizeNN (Image * image, int newWidth, int newHeight)

RLAPI void ImageRotate (Image * image, int degrees)

RLAPI void ImageRotateCCW (Image * image)

RLAPI void ImageRotateCW (Image * image)

RLAPI Image ImageText (const char * text, int fontSize, Color color)

RLAPI Image ImageTextEx (Font font, const char * text, float fontSize, float spacing, Color tint)

RLAPI void ImageToPOT (Image * image, Color fill)

RLAPI void InitAudioDevice (void)

RLAPI void InitWindow (int width, int height, const char * title)

RLAPI bool IsAudioDeviceReady (void)

RLAPI bool IsAudioStreamPlaying (AudioStream stream)

RLAPI bool IsAudioStreamProcessed (AudioStream stream)

RLAPI bool IsAudioStreamValid (AudioStream stream)

RLAPI bool IsCursorHidden (void)

RLAPI bool IsCursorOnScreen (void)

RLAPI bool IsFileDropped (void)

RLAPI bool IsFileExtension (const char * fileName, const char * ext)

RLAPI bool IsFileNameValid (const char * fileName)

RLAPI bool IsFontValid (Font font)

RLAPI bool IsGamepadAvailable (int gamepad)

RLAPI bool IsGamepadButtonDown (int gamepad, int button)

RLAPI bool IsGamepadButtonPressed (int gamepad, int button)

RLAPI bool IsGamepadButtonReleased (int gamepad, int button)

RLAPI bool IsGamepadButtonUp (int gamepad, int button)

RLAPI bool IsGestureDetected (unsigned int gesture)

RLAPI bool IsImageValid (Image image)

RLAPI bool IsKeyDown (int key)

RLAPI bool IsKeyPressed (int key)

RLAPI bool IsKeyPressedRepeat (int key)

RLAPI bool IsKeyReleased (int key)

RLAPI bool IsKeyUp (int key)

RLAPI bool IsMaterialValid (Material material)

RLAPI bool IsModelAnimationValid (Model model, ModelAnimation anim)

RLAPI bool IsModelValid (Model model)

RLAPI bool IsMouseButtonDown (int button)

RLAPI bool IsMouseButtonPressed (int button)

RLAPI bool IsMouseButtonReleased (int button)

RLAPI bool IsMouseButtonUp (int button)

RLAPI bool IsMusicStreamPlaying (Music music)

RLAPI bool IsMusicValid (Music music)

RLAPI bool IsPathFile (const char * path)

RLAPI bool IsRenderTextureValid (RenderTexture2D target)

RLAPI bool IsShaderValid (Shader shader)

RLAPI bool IsSoundPlaying (Sound sound)

RLAPI bool IsSoundValid (Sound sound)

RLAPI bool IsTextureValid (Texture2D texture)

RLAPI bool IsWaveValid (Wave wave)

RLAPI bool IsWindowFocused (void)

RLAPI bool IsWindowFullscreen (void)

RLAPI bool IsWindowHidden (void)

RLAPI bool IsWindowMaximized (void)

RLAPI bool IsWindowMinimized (void)

RLAPI bool IsWindowReady (void)

RLAPI bool IsWindowResized (void)

RLAPI bool IsWindowState (unsigned int flag)

RLAPI AudioStream LoadAudioStream (unsigned int sampleRate, unsigned int sampleSize, unsigned int channels)

RLAPI AutomationEventList LoadAutomationEventList (const char * fileName)

RLAPI int * LoadCodepoints (const char * text, int * count)

RLAPI FilePathList LoadDirectoryFiles (const char * dirPath)

RLAPI FilePathList LoadDirectoryFilesEx (const char * basePath, const char * filter, bool scanSubdirs)

RLAPI FilePathList LoadDroppedFiles (void)

RLAPI unsigned char * LoadFileData (const char * fileName, int * dataSize)

RLAPI char * LoadFileText (const char * fileName)

RLAPI Font LoadFont (const char * fileName)

RLAPI GlyphInfo * LoadFontData (const unsigned char * fileData, int dataSize, int fontSize, int * codepoints, int codepointCount, int type)

RLAPI Font LoadFontEx (const char * fileName, int fontSize, int * codepoints, int codepointCount)

RLAPI Font LoadFontFromImage (Image image, Color key, int firstChar)

RLAPI Font LoadFontFromMemory (const char * fileType, const unsigned char * fileData, int dataSize, int fontSize, int * codepoints, int codepointCount)

RLAPI Image LoadImage (const char * fileName)

RLAPI Image LoadImageAnim (const char * fileName, int * frames)

RLAPI Image LoadImageAnimFromMemory (const char * fileType, const unsigned char * fileData, int dataSize, int * frames)

RLAPI Color * LoadImageColors (Image image)

RLAPI Image LoadImageFromMemory (const char * fileType, const unsigned char * fileData, int dataSize)

RLAPI Image LoadImageFromScreen (void)

RLAPI Image LoadImageFromTexture (Texture2D texture)

RLAPI Color * LoadImagePalette (Image image, int maxPaletteSize, int * colorCount)

RLAPI Image LoadImageRaw (const char * fileName, int width, int height, int format, int headerSize)

RLAPI Material LoadMaterialDefault (void)

RLAPI Material * LoadMaterials (const char * fileName, int * materialCount)

RLAPI Model LoadModel (const char * fileName)

RLAPI ModelAnimation * LoadModelAnimations (const char * fileName, int * animCount)

RLAPI Model LoadModelFromMesh (Mesh mesh)

RLAPI Music LoadMusicStream (const char * fileName)

RLAPI Music LoadMusicStreamFromMemory (const char * fileType, const unsigned char * data, int dataSize)

RLAPI int * LoadRandomSequence (unsigned int count, int min, int max)

RLAPI RenderTexture2D LoadRenderTexture (int width, int height)

RLAPI Shader LoadShader (const char * vsFileName, const char * fsFileName)

RLAPI Shader LoadShaderFromMemory (const char * vsCode, const char * fsCode)

RLAPI Sound LoadSound (const char * fileName)

RLAPI Sound LoadSoundAlias (Sound source)

RLAPI Sound LoadSoundFromWave (Wave wave)

RLAPI Texture2D LoadTexture (const char * fileName)

RLAPI TextureCubemap LoadTextureCubemap (Image image, int layout)

RLAPI Texture2D LoadTextureFromImage (Image image)

RLAPI char * LoadUTF8 (const int * codepoints, int length)

RLAPI VrStereoConfig LoadVrStereoConfig (VrDeviceInfo device)

RLAPI Wave LoadWave (const char * fileName)

RLAPI Wave LoadWaveFromMemory (const char * fileType, const unsigned char * fileData, int dataSize)

RLAPI float * LoadWaveSamples (Wave wave)

RLAPI int MakeDirectory (const char * dirPath)

RLAPI void MaximizeWindow (void)

RLAPI int MeasureText (const char * text, int fontSize)

RLAPI Vector2 MeasureTextEx (Font font, const char * text, float fontSize, float spacing)

RLAPI void * MemAlloc (unsigned int size)

RLAPI void MemFree (void * ptr)

RLAPI void * MemRealloc (void * ptr, unsigned int size)

RLAPI void MinimizeWindow (void)

RLAPI void OpenURL (const char * url)

RLAPI void PauseAudioStream (AudioStream stream)

RLAPI void PauseMusicStream (Music music)

RLAPI void PauseSound (Sound sound)

RLAPI void PlayAudioStream (AudioStream stream)

RLAPI void PlayAutomationEvent (AutomationEvent event)

RLAPI void PlayMusicStream (Music music)

RLAPI void PlaySound (Sound sound)

RLAPI void PollInputEvents (void)

RLAPI void RestoreWindow (void)

RLAPI void ResumeAudioStream (AudioStream stream)

RLAPI void ResumeMusicStream (Music music)

RLAPI void ResumeSound (Sound sound)

RLAPI bool SaveFileData (const char * fileName, void * data, int dataSize)

RLAPI bool SaveFileText (const char * fileName, char * text)

RLAPI void SeekMusicStream (Music music, float position)

RLAPI void SetAudioStreamBufferSizeDefault (int size)

RLAPI void SetAudioStreamCallback (AudioStream stream, AudioCallback callback)

RLAPI void SetAudioStreamPan (AudioStream stream, float pan)

RLAPI void SetAudioStreamPitch (AudioStream stream, float pitch)

RLAPI void SetAudioStreamVolume (AudioStream stream, float volume)

RLAPI void SetAutomationEventBaseFrame (int frame)

RLAPI void SetAutomationEventList (AutomationEventList * list)

RLAPI void SetClipboardText (const char * text)

RLAPI void SetConfigFlags (unsigned int flags)

RLAPI void SetExitKey (int key)

RLAPI int SetGamepadMappings (const char * mappings)

RLAPI void SetGamepadVibration (int gamepad, float leftMotor, float rightMotor, float duration)

RLAPI void SetGesturesEnabled (unsigned int flags)

RLAPI void SetLoadFileDataCallback (LoadFileDataCallback callback)

RLAPI void SetLoadFileTextCallback (LoadFileTextCallback callback)

RLAPI void SetMasterVolume (float volume)

RLAPI void SetMaterialTexture (Material * material, int mapType, Texture2D texture)

RLAPI void SetModelMeshMaterial (Model * model, int meshId, int materialId)

RLAPI void SetMouseCursor (int cursor)

RLAPI void SetMouseOffset (int offsetX, int offsetY)

RLAPI void SetMousePosition (int x, int y)

RLAPI void SetMouseScale (float scaleX, float scaleY)

RLAPI void SetMusicPan (Music music, float pan)

RLAPI void SetMusicPitch (Music music, float pitch)

RLAPI void SetMusicVolume (Music music, float volume)

RLAPI void SetPixelColor (void * dstPtr, Color color, int format)

RLAPI void SetRandomSeed (unsigned int seed)

RLAPI void SetSaveFileDataCallback (SaveFileDataCallback callback)

RLAPI void SetSaveFileTextCallback (SaveFileTextCallback callback)

RLAPI void SetShaderValue (Shader shader, int locIndex, const void * value, int uniformType)

RLAPI void SetShaderValueMatrix (Shader shader, int locIndex, Matrix mat)

RLAPI void SetShaderValueTexture (Shader shader, int locIndex, Texture2D texture)

RLAPI void SetShaderValueV (Shader shader, int locIndex, const void * value, int uniformType, int count)

RLAPI void SetShapesTexture (Texture2D texture, Rectangle source)

RLAPI void SetSoundPan (Sound sound, float pan)

RLAPI void SetSoundPitch (Sound sound, float pitch)

RLAPI void SetSoundVolume (Sound sound, float volume)

RLAPI void SetTargetFPS (int fps)

RLAPI void SetTextLineSpacing (int spacing)

RLAPI void SetTextureFilter (Texture2D texture, int filter)

RLAPI void SetTextureWrap (Texture2D texture, int wrap)

RLAPI void SetTraceLogCallback (TraceLogCallback callback)

RLAPI void SetTraceLogLevel (int logLevel)

RLAPI void SetWindowFocused (void)

RLAPI void SetWindowIcon (Image image)

RLAPI void SetWindowIcons (Image * images, int count)

RLAPI void SetWindowMaxSize (int width, int height)

RLAPI void SetWindowMinSize (int width, int height)

RLAPI void SetWindowMonitor (int monitor)

RLAPI void SetWindowOpacity (float opacity)

RLAPI void SetWindowPosition (int x, int y)

RLAPI void SetWindowSize (int width, int height)

RLAPI void SetWindowState (unsigned int flags)

RLAPI void SetWindowTitle (const char * title)

RLAPI void ShowCursor (void)

RLAPI void StartAutomationEventRecording (void)

RLAPI void StopAudioStream (AudioStream stream)

RLAPI void StopAutomationEventRecording (void)

RLAPI void StopMusicStream (Music music)

RLAPI void StopSound (Sound sound)

RLAPI void SwapScreenBuffer (void)

RLAPI void TakeScreenshot (const char * fileName)

RLAPI void TextAppend (char * text, const char * append, int * position)

RLAPI int TextCopy (char * dst, const char * src)

RLAPI int TextFindIndex (const char * text, const char * find)

RLAPI const char * TextFormat (const char * text, ...)

RLAPI char * TextInsert (const char * text, const char * insert, int position)

RLAPI bool TextIsEqual (const char * text1, const char * text2)

RLAPI const char * TextJoin (const char ** textList, int count, const char * delimiter)

RLAPI unsigned int TextLength (const char * text)

RLAPI char * TextReplace (const char * text, const char * replace, const char * by)

RLAPI const char ** TextSplit (const char * text, char delimiter, int * count)

RLAPI const char * TextSubtext (const char * text, int position, int length)

RLAPI const char * TextToCamel (const char * text)

RLAPI float TextToFloat (const char * text)

RLAPI int TextToInteger (const char * text)

RLAPI const char * TextToLower (const char * text)

RLAPI const char * TextToPascal (const char * text)

RLAPI const char * TextToSnake (const char * text)

RLAPI const char * TextToUpper (const char * text)

RLAPI void ToggleBorderlessWindowed (void)

RLAPI void ToggleFullscreen (void)

RLAPI void TraceLog (int logLevel, const char * text, ...)

RLAPI void UnloadAudioStream (AudioStream stream)

RLAPI void UnloadAutomationEventList (AutomationEventList list)

RLAPI void UnloadCodepoints (int * codepoints)

RLAPI void UnloadDirectoryFiles (FilePathList files)

RLAPI void UnloadDroppedFiles (FilePathList files)

RLAPI void UnloadFileData (unsigned char * data)

RLAPI void UnloadFileText (char * text)

RLAPI void UnloadFont (Font font)

RLAPI void UnloadFontData (GlyphInfo * glyphs, int glyphCount)

RLAPI void UnloadImage (Image image)

RLAPI void UnloadImageColors (Color * colors)

RLAPI void UnloadImagePalette (Color * colors)

RLAPI void UnloadMaterial (Material material)

RLAPI void UnloadMesh (Mesh mesh)

RLAPI void UnloadModel (Model model)

RLAPI void UnloadModelAnimation (ModelAnimation anim)

RLAPI void UnloadModelAnimations (ModelAnimation * animations, int animCount)

RLAPI void UnloadMusicStream (Music music)

RLAPI void UnloadRandomSequence (int * sequence)

RLAPI void UnloadRenderTexture (RenderTexture2D target)

RLAPI void UnloadShader (Shader shader)

RLAPI void UnloadSound (Sound sound)

RLAPI void UnloadSoundAlias (Sound alias)

RLAPI void UnloadTexture (Texture2D texture)

RLAPI void UnloadUTF8 (char * text)

RLAPI void UnloadVrStereoConfig (VrStereoConfig config)

RLAPI void UnloadWave (Wave wave)

RLAPI void UnloadWaveSamples (float * samples)

RLAPI void UpdateAudioStream (AudioStream stream, const void * data, int frameCount)

RLAPI void UpdateCamera (Camera * camera, int mode)

RLAPI void UpdateCameraPro (Camera * camera, Vector3 movement, Vector3 rotation, float zoom)

RLAPI void UpdateMeshBuffer (Mesh mesh, int index, const void * data, int dataSize, int offset)

RLAPI void UpdateModelAnimation (Model model, ModelAnimation anim, int frame)

RLAPI void UpdateModelAnimationBones (Model model, ModelAnimation anim, int frame)

RLAPI void UpdateMusicStream (Music music)

RLAPI void UpdateSound (Sound sound, const void * data, int sampleCount)

RLAPI void UpdateTexture (Texture2D texture, const void * pixels)

RLAPI void UpdateTextureRec (Texture2D texture, Rectangle rec, const void * pixels)

RLAPI void UploadMesh (Mesh * mesh, bool dynamic)

RLAPI void WaitTime (double seconds)

RLAPI Wave WaveCopy (Wave wave)

RLAPI void WaveCrop (Wave * wave, int initFrame, int finalFrame)

RLAPI void WaveFormat (Wave * wave, int sampleRate, int sampleSize, int channels)

RLAPI bool WindowShouldClose (void)

raylib.h

Ir para a documentação desse arquivo.

```c
/**********************************************************************************************
*
*   raylib v5.5 - A simple and easy-to-use library to enjoy videogames programming (www.raylib.com)
*
*   FEATURES:
*       - NO external dependencies, all required libraries included with raylib
*       - Multiplatform: Windows, Linux, FreeBSD, OpenBSD, NetBSD, DragonFly,
*                        MacOS, Haiku, Android, Raspberry Pi, DRM native, HTML5.
*       - Written in plain C code (C99) in PascalCase/camelCase notation
*       - Hardware accelerated with OpenGL (1.1, 2.1, 3.3, 4.3, ES2, ES3 - choose at compile)
*       - Unique OpenGL abstraction layer (usable as standalone module): [rlgl]
*       - Multiple Fonts formats supported (TTF, OTF, FNT, BDF, Sprite fonts)
*       - Outstanding texture formats support, including compressed formats (DXT, ETC, ASTC)
*       - Full 3d support for 3d Shapes, Models, Billboards, Heightmaps and more!
*       - Flexible Materials system, supporting classic maps and PBR maps
*       - Animated 3D models supported (skeletal bones animation) (IQM, M3D, GLTF)
*       - Shaders support, including Model shaders and Postprocessing shaders
*       - Powerful math module for Vector, Matrix and Quaternion operations: [raymath]
*       - Audio loading and playing with streaming support (WAV, OGG, MP3, FLAC, QOA, XM, MOD)
*       - VR stereo rendering with configurable HMD device parameters
*       - Bindings to multiple programming languages available!
*
*   NOTES:
*       - One default Font is loaded on InitWindow()->LoadFontDefault() [core, text]
*       - One default Texture2D is loaded on rlglInit(), 1x1 white pixel R8G8B8A8 [rlgl] (OpenGL 3.3 or ES2)
*       - One default Shader is loaded on rlglInit()->rlLoadShaderDefault() [rlgl] (OpenGL 3.3 or ES2)
*       - One default RenderBatch is loaded on rlglInit()->rlLoadRenderBatch() [rlgl] (OpenGL 3.3 or ES2)
*
*   DEPENDENCIES (included):
*       [rcore][GLFW] rglfw (Camilla Löwy - github.com/glfw/glfw) for window/context management and input
*       [rcore][RGFW] rgfw (ColleagueRiley - github.com/ColleagueRiley/RGFW) for window/context management and input
*       [rlgl] glad/glad_gles2 (David Herberth - github.com/Dav1dde/glad) for OpenGL 3.3 extensions loading
*       [raudio] miniaudio (David Reid - github.com/mackron/miniaudio) for audio device/context management
*
*   OPTIONAL DEPENDENCIES (included):
*       [rcore] msf_gif (Miles Fogle) for GIF recording
*       [rcore] sinfl (Micha Mettke) for DEFLATE decompression algorithm
*       [rcore] sdefl (Micha Mettke) for DEFLATE compression algorithm
*       [rcore] rprand (Ramon Snatamaria) for pseudo-random numbers generation
*       [rtextures] qoi (Dominic Szablewski - https://phoboslab.org) for QOI image manage
*       [rtextures] stb_image (Sean Barret) for images loading (BMP, TGA, PNG, JPEG, HDR...)
*       [rtextures] stb_image_write (Sean Barret) for image writing (BMP, TGA, PNG, JPG)
*       [rtextures] stb_image_resize2 (Sean Barret) for image resizing algorithms
*       [rtextures] stb_perlin (Sean Barret) for Perlin Noise image generation
*       [rtext] stb_truetype (Sean Barret) for ttf fonts loading
*       [rtext] stb_rect_pack (Sean Barret) for rectangles packing
*       [rmodels] par_shapes (Philip Rideout) for parametric 3d shapes generation
*       [rmodels] tinyobj_loader_c (Syoyo Fujita) for models loading (OBJ, MTL)
*       [rmodels] cgltf (Johannes Kuhlmann) for models loading (glTF)
*       [rmodels] m3d (bzt) for models loading (M3D, https://bztsrc.gitlab.io/model3d)
*       [rmodels] vox_loader (Johann Nadalutti) for models loading (VOX)
*       [raudio] dr_wav (David Reid) for WAV audio file loading
*       [raudio] dr_flac (David Reid) for FLAC audio file loading
*       [raudio] dr_mp3 (David Reid) for MP3 audio file loading
*       [raudio] stb_vorbis (Sean Barret) for OGG audio loading
*       [raudio] jar_xm (Joshua Reisenauer) for XM audio module loading
*       [raudio] jar_mod (Joshua Reisenauer) for MOD audio module loading
*       [raudio] qoa (Dominic Szablewski - https://phoboslab.org) for QOA audio manage
*
*
*   LICENSE: zlib/libpng
*
*   raylib is licensed under an unmodified zlib/libpng license, which is an OSI-certified,
*   BSD-like license that allows static linking with closed source software:
*
*   Copyright (c) 2013-2024 Ramon Santamaria (@raysan5)
*
*   This software is provided "as-is", without any express or implied warranty. In no event
*   will the authors be held liable for any damages arising from the use of this software.
*
*   Permission is granted to anyone to use this software for any purpose, including commercial
*   applications, and to alter it and redistribute it freely, subject to the following restrictions:
*
*     1. The origin of this software must not be misrepresented; you must not claim that you
*     wrote the original software. If you use this software in a product, an acknowledgment
*     in the product documentation would be appreciated but is not required.
*
*     2. Altered source versions must be plainly marked as such, and must not be misrepresented
*     as being the original software.
*
*     3. This notice may not be removed or altered from any source distribution.
*
**********************************************************************************************/

#ifndef RAYLIB_H
#define RAYLIB_H

#include <stdarg.h>     // Required for: va_list - Only used by TraceLogCallback

#define RAYLIB_VERSION_MAJOR 5
#define RAYLIB_VERSION_MINOR 5
#define RAYLIB_VERSION_PATCH 0
#define RAYLIB_VERSION  "5.5"

// Function specifiers in case library is build/used as a shared library
// NOTE: Microsoft specifiers to tell compiler that symbols are imported/exported from a .dll
// NOTE: visibility("default") attribute makes symbols "visible" when compiled with -fvisibility=hidden
#if defined(_WIN32)
    #if defined(__TINYC__)
        #define __declspec(x) __attribute__((x))
    #endif
    #if defined(BUILD_LIBTYPE_SHARED)
        #define RLAPI __declspec(dllexport)     // We are building the library as a Win32 shared library (.dll)
    #elif defined(USE_LIBTYPE_SHARED)
        #define RLAPI __declspec(dllimport)     // We are using the library as a Win32 shared library (.dll)
    #endif
#else
    #if defined(BUILD_LIBTYPE_SHARED)
        #define RLAPI __attribute__((visibility("default"))) // We are building as a Unix shared library (.so/.dylib)
    #endif
#endif

#ifndef RLAPI
    #define RLAPI       // Functions defined as 'extern' by default (implicit specifiers)
#endif

```

```
//----------------------------------------------------------------------------------
// Some basic Defines
//----------------------------------------------------------------------------------
#ifndef PI
    #define PI 3.14159265358979323846f
#endif
#ifndef DEG2RAD
    #define DEG2RAD (PI/180.0f)
#endif
#ifndef RAD2DEG
    #define RAD2DEG (180.0f/PI)
#endif

// Allow custom memory allocators
// NOTE: Require recompiling raylib sources
#ifndef RL_MALLOC
    #define RL_MALLOC(sz)       malloc(sz)
#endif
#ifndef RL_CALLOC
    #define RL_CALLOC(n,sz)     calloc(n,sz)
#endif
#ifndef RL_REALLOC
    #define RL_REALLOC(ptr,sz)  realloc(ptr,sz)
#endif
#ifndef RL_FREE
    #define RL_FREE(ptr)        free(ptr)
#endif

// NOTE: MSVC C++ compiler does not support compound literals (C99 feature)
// Plain structures in C++ (without constructors) can be initialized with { }
// This is called aggregate initialization (C++11 feature)
#if defined(__cplusplus)
    #define CLITERAL(type)      type
#else
    #define CLITERAL(type)      (type)
#endif

// Some compilers (mostly macos clang) default to C++98,
// where aggregate initialization can't be used
// So, give a more clear error stating how to fix this
#if !defined(_MSC_VER) && (defined(__cplusplus) && __cplusplus < 201103L)
    #error "C++11 or later is required. Add -std=c++11"
#endif

// NOTE: We set some defines with some data types declared by raylib
// Other modules (raymath, rlgl) also require some of those types, so,
// to be able to use those other modules as standalone (not depending on raylib)
// this defines are very useful for internal check and avoid type (re)definitions
#define RL_COLOR_TYPE
#define RL_RECTANGLE_TYPE
#define RL_VECTOR2_TYPE
#define RL_VECTOR3_TYPE
#define RL_VECTOR4_TYPE
#define RL_QUATERNION_TYPE
#define RL_MATRIX_TYPE

// Some Basic Colors
// NOTE: Custom raylib color palette for amazing visuals on WHITE background
#define LIGHTGRAY  CLITERAL(Color){ 200, 200, 200, 255 }   // Light Gray
#define GRAY       CLITERAL(Color){ 130, 130, 130, 255 }   // Gray
#define DARKGRAY   CLITERAL(Color){ 80, 80, 80, 255 }      // Dark Gray
#define YELLOW     CLITERAL(Color){ 253, 249, 0, 255 }     // Yellow
#define GOLD       CLITERAL(Color){ 255, 203, 0, 255 }     // Gold
#define ORANGE     CLITERAL(Color){ 255, 161, 0, 255 }     // Orange
#define PINK       CLITERAL(Color){ 255, 109, 194, 255 }   // Pink
#define RED        CLITERAL(Color){ 230, 41, 55, 255 }     // Red
#define MAROON     CLITERAL(Color){ 190, 33, 55, 255 }     // Maroon
#define GREEN      CLITERAL(Color){ 0, 228, 48, 255 }      // Green
#define LIME       CLITERAL(Color){ 0, 158, 47, 255 }      // Lime
#define DARKGREEN  CLITERAL(Color){ 0, 117, 44, 255 }      // Dark Green
#define SKYBLUE    CLITERAL(Color){ 102, 191, 255, 255 }   // Sky Blue
#define BLUE       CLITERAL(Color){ 0, 121, 241, 255 }     // Blue
#define DARKBLUE   CLITERAL(Color){ 0, 82, 172, 255 }      // Dark Blue
#define PURPLE     CLITERAL(Color){ 200, 122, 255, 255 }   // Purple
#define VIOLET     CLITERAL(Color){ 135, 60, 190, 255 }    // Violet
```

```c
#define DARKPURPLE CLITERAL(Color){ 112, 31, 126, 255 }    // Dark Purple
#define BEIGE      CLITERAL(Color){ 211, 176, 131, 255 }   // Beige
#define BROWN      CLITERAL(Color){ 127, 106, 79, 255 }    // Brown
#define DARKBROWN  CLITERAL(Color){ 76, 63, 47, 255 }      // Dark Brown

#define WHITE      CLITERAL(Color){ 255, 255, 255, 255 }   // White
#define BLACK      CLITERAL(Color){ 0, 0, 0, 255 }         // Black
#define BLANK      CLITERAL(Color){ 0, 0, 0, 0 }           // Blank (Transparent)
#define MAGENTA    CLITERAL(Color){ 255, 0, 255, 255 }     // Magenta
#define RAYWHITE   CLITERAL(Color){ 245, 245, 245, 255 }   // My own White (raylib logo)


//----------------------------------------------------------------------------------
// Structures Definition
//----------------------------------------------------------------------------------
// Boolean type
#if (defined(__STDC__) && __STDC_VERSION__ >= 199901L) || (defined(_MSC_VER) && _MSC_VER >= 1800)
    #include <stdbool.h>
#elif !defined(__cplusplus) && !defined(bool)
    typedef enum bool { false = 0, true = !false } bool;
    #define RL_BOOL_TYPE
#endif

// Vector2, 2 components
typedef struct Vector2 {
    float x;                // Vector x component
    float y;                // Vector y component
} Vector2;

// Vector3, 3 components
typedef struct Vector3 {
    float x;                // Vector x component
    float y;                // Vector y component
    float z;                // Vector z component
} Vector3;

// Vector4, 4 components
typedef struct Vector4 {
    float x;                // Vector x component
    float y;                // Vector y component
    float z;                // Vector z component
    float w;                // Vector w component
} Vector4;

// Quaternion, 4 components (Vector4 alias)
typedef Vector4 Quaternion;

// Matrix, 4x4 components, column major, OpenGL style, right-handed
typedef struct Matrix {
    float m0, m4, m8, m12;  // Matrix first row (4 components)
    float m1, m5, m9, m13;  // Matrix second row (4 components)
    float m2, m6, m10, m14; // Matrix third row (4 components)
    float m3, m7, m11, m15; // Matrix fourth row (4 components)
} Matrix;

// Color, 4 components, R8G8B8A8 (32bit)
typedef struct Color {
    unsigned char r;        // Color red value
    unsigned char g;        // Color green value
    unsigned char b;        // Color blue value
    unsigned char a;        // Color alpha value
} Color;

// Rectangle, 4 components
typedef struct Rectangle {
    float x;                // Rectangle top-left corner position x
    float y;                // Rectangle top-left corner position y
    float width;            // Rectangle width
    float height;           // Rectangle height
} Rectangle;

// Image, pixel data stored in CPU memory (RAM)
typedef struct Image {
    void *data;             // Image raw data
    int width;              // Image base width
    int height;             // Image base height
    int mipmaps;            // Mipmap levels, 1 by default
    int format;             // Data format (PixelFormat type)
} Image;

// Texture, tex data stored in GPU memory (VRAM)
typedef struct Texture {
    unsigned int id;        // OpenGL texture id
    int width;              // Texture base width
    int height;             // Texture base height
    int mipmaps;            // Mipmap levels, 1 by default
    int format;             // Data format (PixelFormat type)
} Texture;

// Texture2D, same as Texture
typedef Texture Texture2D;

// TextureCubemap, same as Texture
typedef Texture TextureCubemap;

// RenderTexture, fbo for texture rendering
typedef struct RenderTexture {
    unsigned int id;        // OpenGL framebuffer object id
    Texture texture;        // Color buffer attachment texture
    Texture depth;          // Depth buffer attachment texture
} RenderTexture;

// RenderTexture2D, same as RenderTexture
typedef RenderTexture RenderTexture2D;

// NPatchInfo, n-patch layout info
typedef struct NPatchInfo {
    Rectangle source;       // Texture source rectangle
    int left;               // Left border offset
    int top;                // Top border offset
    int right;              // Right border offset
    int bottom;             // Bottom border offset
    int layout;             // Layout of the n-patch: 3x3, 1x3 or 3x1
} NPatchInfo;

// GlyphInfo, font characters glyphs info
typedef struct GlyphInfo {
    int value;              // Character value (Unicode)
    int offsetX;            // Character offset X when drawing
    int offsetY;            // Character offset Y when drawing
    int advanceX;           // Character advance position X
    Image image;            // Character image data
} GlyphInfo;

// Font, font texture and GlyphInfo array data
typedef struct Font {
    int baseSize;           // Base size (default chars height)
    int glyphCount;         // Number of glyph characters
    int glyphPadding;       // Padding around the glyph characters
    Texture2D texture;      // Texture atlas containing the glyphs
    Rectangle *recs;        // Rectangles in texture for the glyphs
    GlyphInfo *glyphs;      // Glyphs info data
} Font;

// Camera, defines position/orientation in 3d space
typedef struct Camera3D {
    Vector3 position;       // Camera position
    Vector3 target;         // Camera target it looks-at
    Vector3 up;             // Camera up vector (rotation over its axis)
    float fovy;             // Camera field-of-view aperture in Y (degrees) in perspective, used as near plane width in orthographic
    int projection;         // Camera projection: CAMERA_PERSPECTIVE or CAMERA_ORTHOGRAPHIC
} Camera3D;

typedef Camera3D Camera;    // Camera type fallback, defaults to Camera3D

// Camera2D, defines position/orientation in 2d space
typedef struct Camera2D {
    Vector2 offset;         // Camera offset (displacement from target)
    Vector2 target;         // Camera target (rotation and zoom origin)
    float rotation;         // Camera rotation in degrees
    float zoom;             // Camera zoom (scaling), should be 1.0f by default
} Camera2D;

// Mesh, vertex data and vao/vbo
typedef struct Mesh {
    int vertexCount;        // Number of vertices stored in arrays
    int triangleCount;      // Number of triangles stored (indexed or not)

    // Vertex attributes data
    float *vertices;        // Vertex position (XYZ - 3 components per vertex) (shader-location = 0)
    float *texcoords;       // Vertex texture coordinates (UV - 2 components per vertex) (shader-location = 1)
    float *texcoords2;      // Vertex texture second coordinates (UV - 2 components per vertex) (shader-location = 5)
    float *normals;         // Vertex normals (XYZ - 3 components per vertex) (shader-location = 2)
    float *tangents;        // Vertex tangents (XYZW - 4 components per vertex) (shader-location = 4)
    unsigned char *colors;      // Vertex colors (RGBA - 4 components per vertex) (shader-location = 3)
    unsigned short *indices;    // Vertex indices (in case vertex data comes indexed)

    // Animation vertex data
    float *animVertices;    // Animated vertex positions (after bones transformations)
    float *animNormals;     // Animated normals (after bones transformations)
    unsigned char *boneIds; // Vertex bone ids, max 255 bone ids, up to 4 bones influence by vertex (skinning) (shader-location = 6)
    float *boneWeights;     // Vertex bone weight, up to 4 bones influence by vertex (skinning) (shader-location = 7)
    Matrix *boneMatrices;   // Bones animated transformation matrices
    int boneCount;          // Number of bones

    // OpenGL identifiers
    unsigned int vaoId;     // OpenGL Vertex Array Object id
    unsigned int *vboId;    // OpenGL Vertex Buffer Objects id (default vertex data)
} Mesh;

// Shader
typedef struct Shader {
    unsigned int id;        // Shader program id
    int *locs;              // Shader locations array (RL_MAX_SHADER_LOCATIONS)
} Shader;

// MaterialMap
typedef struct MaterialMap {
    Texture2D texture;      // Material map texture
    Color color;            // Material map color
    float value;            // Material map value
} MaterialMap;

// Material, includes shader and maps
typedef struct Material {
    Shader shader;          // Material shader
    MaterialMap *maps;      // Material maps array (MAX_MATERIAL_MAPS)
    float params[4];        // Material generic parameters (if required)
} Material;

// Transform, vertex transformation data
typedef struct Transform {
    Vector3 translation;    // Translation
    Quaternion rotation;    // Rotation
    Vector3 scale;          // Scale
} Transform;

// Bone, skeletal animation bone
typedef struct BoneInfo {
    char name[32];          // Bone name
    int parent;             // Bone parent
} BoneInfo;

// Model, meshes, materials and animation data
typedef struct Model {
    Matrix transform;       // Local transform matrix

    int meshCount;          // Number of meshes
    int materialCount;      // Number of materials
    Mesh *meshes;           // Meshes array
    Material *materials;    // Materials array
    int *meshMaterial;      // Mesh material number

    // Animation data
    int boneCount;          // Number of bones
    BoneInfo *bones;        // Bones information (skeleton)
    Transform *bindPose;    // Bones base transformation (pose)
} Model;

// ModelAnimation
typedef struct ModelAnimation {
    int boneCount;          // Number of bones
    int frameCount;         // Number of animation frames
    BoneInfo *bones;        // Bones information (skeleton)
    Transform **framePoses; // Poses array by frame
    char name[32];          // Animation name
} ModelAnimation;

// Ray, ray for raycasting
typedef struct Ray {
    Vector3 position;       // Ray position (origin)
    Vector3 direction;      // Ray direction (normalized)
} Ray;

// RayCollision, ray hit information
typedef struct RayCollision {
    bool hit;               // Did the ray hit something?
    float distance;         // Distance to the nearest hit
    Vector3 point;          // Point of the nearest hit
    Vector3 normal;         // Surface normal of hit
} RayCollision;

// BoundingBox
typedef struct BoundingBox {
    Vector3 min;            // Minimum vertex box-corner
    Vector3 max;            // Maximum vertex box-corner
} BoundingBox;

// Wave, audio wave data
typedef struct Wave {
    unsigned int frameCount;    // Total number of frames (considering channels)
    unsigned int sampleRate;    // Frequency (samples per second)
    unsigned int sampleSize;    // Bit depth (bits per sample): 8, 16, 32 (24 not
supported)
    unsigned int channels;      // Number of channels (1-mono, 2-stereo, ...)
    void *data;                 // Buffer data pointer
} Wave;

// Opaque structs declaration
// NOTE: Actual structs are defined internally in raudio module
typedef struct rAudioBuffer rAudioBuffer;
typedef struct rAudioProcessor rAudioProcessor;

// AudioStream, custom audio stream
typedef struct AudioStream {
    rAudioBuffer *buffer;       // Pointer to internal data used by the audio system
    rAudioProcessor *processor; // Pointer to internal data processor, useful for audio
effects

    unsigned int sampleRate;    // Frequency (samples per second)
    unsigned int sampleSize;    // Bit depth (bits per sample): 8, 16, 32 (24 not
supported)
    unsigned int channels;      // Number of channels (1-mono, 2-stereo, ...)
} AudioStream;

// Sound
typedef struct Sound {
    AudioStream stream;         // Audio stream
    unsigned int frameCount;    // Total number of frames (considering channels)
} Sound;

// Music, audio stream, anything longer than ~10 seconds should be streamed
typedef struct Music {
    AudioStream stream;         // Audio stream
    unsigned int frameCount;    // Total number of frames (considering channels)
```

```c
    bool looping;               // Music looping enable

    int ctxType;                // Type of music context (audio filetype)
    void *ctxData;              // Audio context data, depends on type
} Music;

// VrDeviceInfo, Head-Mounted-Display device parameters
typedef struct VrDeviceInfo {
    int hResolution;                // Horizontal resolution in pixels
    int vResolution;                // Vertical resolution in pixels
    float hScreenSize;              // Horizontal size in meters
    float vScreenSize;              // Vertical size in meters
    float eyeToScreenDistance;      // Distance between eye and display in meters
    float lensSeparationDistance;   // Lens separation distance in meters
    float interpupillaryDistance;   // IPD (distance between pupils) in meters
    float lensDistortionValues[4];  // Lens distortion constant parameters
    float chromaAbCorrection[4];    // Chromatic aberration correction parameters
} VrDeviceInfo;

// VrStereoConfig, VR stereo rendering configuration for simulator
typedef struct VrStereoConfig {
    Matrix projection[2];           // VR projection matrices (per eye)
    Matrix viewOffset[2];           // VR view offset matrices (per eye)
    float leftLensCenter[2];        // VR left lens center
    float rightLensCenter[2];       // VR right lens center
    float leftScreenCenter[2];      // VR left screen center
    float rightScreenCenter[2];     // VR right screen center
    float scale[2];                 // VR distortion scale
    float scaleIn[2];               // VR distortion scale in
} VrStereoConfig;

// File path list
typedef struct FilePathList {
    unsigned int capacity;          // Filepaths max entries
    unsigned int count;             // Filepaths entries count
    char **paths;                   // Filepaths entries
} FilePathList;

// Automation event
typedef struct AutomationEvent {
    unsigned int frame;             // Event frame
    unsigned int type;              // Event type (AutomationEventType)
    int params[4];                  // Event parameters (if required)
} AutomationEvent;

// Automation event list
typedef struct AutomationEventList {
    unsigned int capacity;          // Events max entries (MAX_AUTOMATION_EVENTS)
    unsigned int count;             // Events entries count
    AutomationEvent *events;        // Events entries
} AutomationEventList;


//----------------------------------------------------------------------------------
// Enumerators Definition

//----------------------------------------------------------------------------------
// System/Window config flags
// NOTE: Every bit registers one state (use it with bit masks)
// By default all flags are set to 0
typedef enum {
    FLAG_VSYNC_HINT         = 0x00000040,   // Set to try enabling V-Sync on GPU
    FLAG_FULLSCREEN_MODE    = 0x00000002,   // Set to run program in fullscreen
    FLAG_WINDOW_RESIZABLE   = 0x00000004,   // Set to allow resizable window
    FLAG_WINDOW_UNDECORATED = 0x00000008,   // Set to disable window decoration (frame and buttons)
    FLAG_WINDOW_HIDDEN      = 0x00000080,   // Set to hide window
    FLAG_WINDOW_MINIMIZED   = 0x00000200,   // Set to minimize window (iconify)
    FLAG_WINDOW_MAXIMIZED   = 0x00000400,   // Set to maximize window (expanded to monitor)
    FLAG_WINDOW_UNFOCUSED   = 0x00000800,   // Set to window non focused
    FLAG_WINDOW_TOPMOST     = 0x00001000,   // Set to window always on top
    FLAG_WINDOW_ALWAYS_RUN  = 0x00000100,   // Set to allow windows running while minimized
    FLAG_WINDOW_TRANSPARENT = 0x00000010,   // Set to allow transparent framebuffer
    FLAG_WINDOW_HIGHDPI     = 0x00002000,   // Set to support HighDPI
    FLAG_WINDOW_MOUSE_PASSTHROUGH = 0x00004000, // Set to support mouse passthrough, only supported when FLAG_WINDOW_UNDECORATED
    FLAG_BORDERLESS_WINDOWED_MODE = 0x00008000, // Set to run program in borderless windowed mode
    FLAG_MSAA_4X_HINT       = 0x00000020,   // Set to try enabling MSAA 4X
    FLAG_INTERLACED_HINT    = 0x00010000    // Set to try enabling interlaced video format (for V3D)
} ConfigFlags;

// Trace log level
// NOTE: Organized by priority level
typedef enum {
    LOG_ALL = 0,        // Display all logs
    LOG_TRACE,          // Trace logging, intended for internal use only
    LOG_DEBUG,          // Debug logging, used for internal debugging, it should be disabled on release builds
    LOG_INFO,           // Info logging, used for program execution info
    LOG_WARNING,        // Warning logging, used on recoverable failures
    LOG_ERROR,          // Error logging, used on unrecoverable failures
    LOG_FATAL,          // Fatal logging, used to abort program: exit(EXIT_FAILURE)
    LOG_NONE            // Disable logging
} TraceLogLevel;

// Keyboard keys (US keyboard layout)
// NOTE: Use GetKeyPressed() to allow redefining
// required keys for alternative layouts
typedef enum {
    KEY_NULL            = 0,        // Key: NULL, used for no key pressed
    // Alphanumeric keys
    KEY_APOSTROPHE      = 39,       // Key: '
    KEY_COMMA           = 44,       // Key: ,
    KEY_MINUS           = 45,       // Key: -
    KEY_PERIOD          = 46,       // Key: .
    KEY_SLASH           = 47,       // Key: /
    KEY_ZERO            = 48,       // Key: 0
    KEY_ONE             = 49,       // Key: 1
    KEY_TWO             = 50,       // Key: 2
    KEY_THREE           = 51,       // Key: 3
    KEY_FOUR            = 52,       // Key: 4
    KEY_FIVE            = 53,       // Key: 5
    KEY_SIX             = 54,       // Key: 6
    KEY_SEVEN           = 55,       // Key: 7
    KEY_EIGHT           = 56,       // Key: 8
    KEY_NINE            = 57,       // Key: 9
    KEY_SEMICOLON       = 59,       // Key: ;
    KEY_EQUAL           = 61,       // Key: =
    KEY_A               = 65,       // Key: A | a
    KEY_B               = 66,       // Key: B | b
    KEY_C               = 67,       // Key: C | c
    KEY_D               = 68,       // Key: D | d
    KEY_E               = 69,       // Key: E | e
    KEY_F               = 70,       // Key: F | f
    KEY_G               = 71,       // Key: G | g
    KEY_H               = 72,       // Key: H | h
    KEY_I               = 73,       // Key: I | i
    KEY_J               = 74,       // Key: J | j
    KEY_K               = 75,       // Key: K | k
    KEY_L               = 76,       // Key: L | l
    KEY_M               = 77,       // Key: M | m
    KEY_N               = 78,       // Key: N | n
    KEY_O               = 79,       // Key: O | o
    KEY_P               = 80,       // Key: P | p
    KEY_Q               = 81,       // Key: Q | q
    KEY_R               = 82,       // Key: R | r
    KEY_S               = 83,       // Key: S | s
    KEY_T               = 84,       // Key: T | t
    KEY_U               = 85,       // Key: U | u
    KEY_V               = 86,       // Key: V | v
    KEY_W               = 87,       // Key: W | w
    KEY_X               = 88,       // Key: X | x
    KEY_Y               = 89,       // Key: Y | y
    KEY_Z               = 90,       // Key: Z | z
    KEY_LEFT_BRACKET    = 91,       // Key: [
    KEY_BACKSLASH       = 92,       // Key: '\'
    KEY_RIGHT_BRACKET   = 93,       // Key: ]
    KEY_GRAVE           = 96,       // Key: `
    // Function keys
    KEY_SPACE           = 32,       // Key: Space
    KEY_ESCAPE          = 256,      // Key: Esc
    KEY_ENTER           = 257,      // Key: Enter
    KEY_TAB             = 258,      // Key: Tab
    KEY_BACKSPACE       = 259,      // Key: Backspace
    KEY_INSERT          = 260,      // Key: Ins
    KEY_DELETE          = 261,      // Key: Del
    KEY_RIGHT           = 262,      // Key: Cursor right
    KEY_LEFT            = 263,      // Key: Cursor left
    KEY_DOWN            = 264,      // Key: Cursor down
    KEY_UP              = 265,      // Key: Cursor up
    KEY_PAGE_UP         = 266,      // Key: Page up
    KEY_PAGE_DOWN       = 267,      // Key: Page down
    KEY_HOME            = 268,      // Key: Home
    KEY_END             = 269,      // Key: End
    KEY_CAPS_LOCK       = 280,      // Key: Caps lock
    KEY_SCROLL_LOCK     = 281,      // Key: Scroll down
    KEY_NUM_LOCK        = 282,      // Key: Num lock
    KEY_PRINT_SCREEN    = 283,      // Key: Print screen
    KEY_PAUSE           = 284,      // Key: Pause
    KEY_F1              = 290,      // Key: F1
    KEY_F2              = 291,      // Key: F2
    KEY_F3              = 292,      // Key: F3
    KEY_F4              = 293,      // Key: F4
    KEY_F5              = 294,      // Key: F5
    KEY_F6              = 295,      // Key: F6
    KEY_F7              = 296,      // Key: F7
    KEY_F8              = 297,      // Key: F8
    KEY_F9              = 298,      // Key: F9
    KEY_F10             = 299,      // Key: F10
    KEY_F11             = 300,      // Key: F11
    KEY_F12             = 301,      // Key: F12
    KEY_LEFT_SHIFT      = 340,      // Key: Shift left
    KEY_LEFT_CONTROL    = 341,      // Key: Control left
    KEY_LEFT_ALT        = 342,      // Key: Alt left
    KEY_LEFT_SUPER      = 343,      // Key: Super left
    KEY_RIGHT_SHIFT     = 344,      // Key: Shift right
    KEY_RIGHT_CONTROL   = 345,      // Key: Control right
    KEY_RIGHT_ALT       = 346,      // Key: Alt right
    KEY_RIGHT_SUPER     = 347,      // Key: Super right
    KEY_KB_MENU         = 348,      // Key: KB menu
    // Keypad keys
    KEY_KP_0            = 320,      // Key: Keypad 0
    KEY_KP_1            = 321,      // Key: Keypad 1
    KEY_KP_2            = 322,      // Key: Keypad 2
    KEY_KP_3            = 323,      // Key: Keypad 3
    KEY_KP_4            = 324,      // Key: Keypad 4
    KEY_KP_5            = 325,      // Key: Keypad 5
    KEY_KP_6            = 326,      // Key: Keypad 6
    KEY_KP_7            = 327,      // Key: Keypad 7
    KEY_KP_8            = 328,      // Key: Keypad 8
    KEY_KP_9            = 329,      // Key: Keypad 9
    KEY_KP_DECIMAL      = 330,      // Key: Keypad .
    KEY_KP_DIVIDE       = 331,      // Key: Keypad /
    KEY_KP_MULTIPLY     = 332,      // Key: Keypad *
    KEY_KP_SUBTRACT     = 333,      // Key: Keypad -
    KEY_KP_ADD          = 334,      // Key: Keypad +
    KEY_KP_ENTER        = 335,      // Key: Keypad Enter
    KEY_KP_EQUAL        = 336,      // Key: Keypad =
    // Android key buttons
    KEY_BACK            = 4,        // Key: Android back button
    KEY_MENU            = 5,        // Key: Android menu button
    KEY_VOLUME_UP       = 24,       // Key: Android volume up button
    KEY_VOLUME_DOWN     = 25        // Key: Android volume down button
} KeyboardKey;

// Add backwards compatibility support for deprecated names
#define MOUSE_LEFT_BUTTON   MOUSE_BUTTON_LEFT
#define MOUSE_RIGHT_BUTTON  MOUSE_BUTTON_RIGHT
#define MOUSE_MIDDLE_BUTTON MOUSE_BUTTON_MIDDLE

// Mouse buttons
typedef enum {
    MOUSE_BUTTON_LEFT    = 0,       // Mouse button left
    MOUSE_BUTTON_RIGHT   = 1,       // Mouse button right
    MOUSE_BUTTON_MIDDLE  = 2,       // Mouse button middle (pressed wheel)
    MOUSE_BUTTON_SIDE    = 3,       // Mouse button side (advanced mouse device)
    MOUSE_BUTTON_EXTRA   = 4,       // Mouse button extra (advanced mouse device)
    MOUSE_BUTTON_FORWARD = 5,       // Mouse button forward (advanced mouse device)
    MOUSE_BUTTON_BACK    = 6,       // Mouse button back (advanced mouse device)
} MouseButton;

// Mouse cursor
typedef enum {
    MOUSE_CURSOR_DEFAULT       = 0,     // Default pointer shape
    MOUSE_CURSOR_ARROW         = 1,     // Arrow shape
    MOUSE_CURSOR_IBEAM         = 2,     // Text writing cursor shape
    MOUSE_CURSOR_CROSSHAIR     = 3,     // Cross shape
    MOUSE_CURSOR_POINTING_HAND = 4,     // Pointing hand cursor
    MOUSE_CURSOR_RESIZE_EW     = 5,     // Horizontal resize/move arrow shape
    MOUSE_CURSOR_RESIZE_NS     = 6,     // Vertical resize/move arrow shape
    MOUSE_CURSOR_RESIZE_NWSE   = 7,     // Top-left to bottom-right diagonal resize/move arrow shape
    MOUSE_CURSOR_RESIZE_NESW   = 8,     // The top-right to bottom-left diagonal resize/move arrow shape
    MOUSE_CURSOR_RESIZE_ALL    = 9,     // The omnidirectional resize/move cursor shape
    MOUSE_CURSOR_NOT_ALLOWED   = 10     // The operation-not-allowed shape
} MouseCursor;

// Gamepad buttons
typedef enum {
    GAMEPAD_BUTTON_UNKNOWN = 0,         // Unknown button, just for error checking
    GAMEPAD_BUTTON_LEFT_FACE_UP,        // Gamepad left DPAD up button
    GAMEPAD_BUTTON_LEFT_FACE_RIGHT,     // Gamepad left DPAD right button
    GAMEPAD_BUTTON_LEFT_FACE_DOWN,      // Gamepad left DPAD down button
    GAMEPAD_BUTTON_LEFT_FACE_LEFT,      // Gamepad left DPAD left button
    GAMEPAD_BUTTON_RIGHT_FACE_UP,       // Gamepad right button up (i.e. PS3: Triangle, Xbox: Y)
    GAMEPAD_BUTTON_RIGHT_FACE_RIGHT,    // Gamepad right button right (i.e. PS3: Circle, Xbox: B)
    GAMEPAD_BUTTON_RIGHT_FACE_DOWN,     // Gamepad right button down (i.e. PS3: Cross, Xbox: A)
    GAMEPAD_BUTTON_RIGHT_FACE_LEFT,     // Gamepad right button left (i.e. PS3: Square, Xbox: X)
    GAMEPAD_BUTTON_LEFT_TRIGGER_1,      // Gamepad top/back trigger left (first), it could be a trailing button
    GAMEPAD_BUTTON_LEFT_TRIGGER_2,      // Gamepad top/back trigger left (second), it could be a trailing button
    GAMEPAD_BUTTON_RIGHT_TRIGGER_1,     // Gamepad top/back trigger right (first), it could be a trailing button
    GAMEPAD_BUTTON_RIGHT_TRIGGER_2,     // Gamepad top/back trigger right (second), it could be a trailing button
    GAMEPAD_BUTTON_MIDDLE_LEFT,         // Gamepad center buttons, left one (i.e. PS3: Select)
    GAMEPAD_BUTTON_MIDDLE,              // Gamepad center buttons, middle one (i.e. PS3: PS, Xbox: XBOX)
    GAMEPAD_BUTTON_MIDDLE_RIGHT,        // Gamepad center buttons, right one (i.e. PS3: Start)
    GAMEPAD_BUTTON_LEFT_THUMB,          // Gamepad joystick pressed button left
    GAMEPAD_BUTTON_RIGHT_THUMB          // Gamepad joystick pressed button right
} GamepadButton;

// Gamepad axis
typedef enum {
    GAMEPAD_AXIS_LEFT_X        = 0,     // Gamepad left stick X axis
    GAMEPAD_AXIS_LEFT_Y        = 1,     // Gamepad left stick Y axis
    GAMEPAD_AXIS_RIGHT_X       = 2,     // Gamepad right stick X axis
    GAMEPAD_AXIS_RIGHT_Y       = 3,     // Gamepad right stick Y axis
    GAMEPAD_AXIS_LEFT_TRIGGER  = 4,     // Gamepad back trigger left, pressure level: [1..-1]
    GAMEPAD_AXIS_RIGHT_TRIGGER = 5      // Gamepad back trigger right, pressure level: [1..-1]
} GamepadAxis;

// Material map index
typedef enum {
    MATERIAL_MAP_ALBEDO = 0,        // Albedo material (same as: MATERIAL_MAP_DIFFUSE)
    MATERIAL_MAP_METALNESS,         // Metalness material (same as: MATERIAL_MAP_SPECULAR)
    MATERIAL_MAP_NORMAL,            // Normal material
    MATERIAL_MAP_ROUGHNESS,         // Roughness material
    MATERIAL_MAP_OCCLUSION,         // Ambient occlusion material
```

```c
    MATERIAL_MAP_EMISSION,          // Emission material
    MATERIAL_MAP_HEIGHT,            // Heightmap material
    MATERIAL_MAP_CUBEMAP,           // Cubemap material (NOTE: Uses GL_TEXTURE_CUBE_MAP)
    MATERIAL_MAP_IRRADIANCE,        // Irradiance material (NOTE: Uses GL_TEXTURE_CUBE_MAP)
    MATERIAL_MAP_PREFILTER,         // Prefilter material (NOTE: Uses GL_TEXTURE_CUBE_MAP)
    MATERIAL_MAP_BRDF               // Brdf material
} MaterialMapIndex;

#define MATERIAL_MAP_DIFFUSE      MATERIAL_MAP_ALBEDO
#define MATERIAL_MAP_SPECULAR     MATERIAL_MAP_METALNESS

// Shader location index
typedef enum {
    SHADER_LOC_VERTEX_POSITION = 0, // Shader location: vertex attribute: position
    SHADER_LOC_VERTEX_TEXCOORD01,   // Shader location: vertex attribute: texcoord01
    SHADER_LOC_VERTEX_TEXCOORD02,   // Shader location: vertex attribute: texcoord02
    SHADER_LOC_VERTEX_NORMAL,       // Shader location: vertex attribute: normal
    SHADER_LOC_VERTEX_TANGENT,      // Shader location: vertex attribute: tangent
    SHADER_LOC_VERTEX_COLOR,        // Shader location: vertex attribute: color
    SHADER_LOC_MATRIX_MVP,          // Shader location: matrix uniform: model-view-projection
    SHADER_LOC_MATRIX_VIEW,         // Shader location: matrix uniform: view (camera transform)
    SHADER_LOC_MATRIX_PROJECTION,   // Shader location: matrix uniform: projection
    SHADER_LOC_MATRIX_MODEL,        // Shader location: matrix uniform: model (transform)
    SHADER_LOC_MATRIX_NORMAL,       // Shader location: matrix uniform: normal
    SHADER_LOC_VECTOR_VIEW,         // Shader location: vector uniform: view
    SHADER_LOC_COLOR_DIFFUSE,       // Shader location: vector uniform: diffuse color
    SHADER_LOC_COLOR_SPECULAR,      // Shader location: vector uniform: specular color
    SHADER_LOC_COLOR_AMBIENT,       // Shader location: vector uniform: ambient color
    SHADER_LOC_MAP_ALBEDO,          // Shader location: sampler2d texture: albedo (same as: SHADER_LOC_MAP_DIFFUSE)
    SHADER_LOC_MAP_METALNESS,       // Shader location: sampler2d texture: metalness (same as: SHADER_LOC_MAP_SPECULAR)
    SHADER_LOC_MAP_NORMAL,          // Shader location: sampler2d texture: normal
    SHADER_LOC_MAP_ROUGHNESS,       // Shader location: sampler2d texture: roughness
    SHADER_LOC_MAP_OCCLUSION,       // Shader location: sampler2d texture: occlusion
    SHADER_LOC_MAP_EMISSION,        // Shader location: sampler2d texture: emission
    SHADER_LOC_MAP_HEIGHT,          // Shader location: sampler2d texture: height
    SHADER_LOC_MAP_CUBEMAP,         // Shader location: samplerCube texture: cubemap
    SHADER_LOC_MAP_IRRADIANCE,      // Shader location: samplerCube texture: irradiance
    SHADER_LOC_MAP_PREFILTER,       // Shader location: samplerCube texture: prefilter
    SHADER_LOC_MAP_BRDF,            // Shader location: sampler2d texture: brdf
    SHADER_LOC_VERTEX_BONEIDS,      // Shader location: vertex attribute: boneIds
    SHADER_LOC_VERTEX_BONEWEIGHTS,  // Shader location: vertex attribute: boneWeights
    SHADER_LOC_BONE_MATRICES        // Shader location: array of matrices uniform: boneMatrices
} ShaderLocationIndex;

#define SHADER_LOC_MAP_DIFFUSE      SHADER_LOC_MAP_ALBEDO
#define SHADER_LOC_MAP_SPECULAR     SHADER_LOC_MAP_METALNESS

// Shader uniform data type
typedef enum {
    SHADER_UNIFORM_FLOAT = 0,       // Shader uniform type: float
    SHADER_UNIFORM_VEC2,            // Shader uniform type: vec2 (2 float)
    SHADER_UNIFORM_VEC3,            // Shader uniform type: vec3 (3 float)
    SHADER_UNIFORM_VEC4,            // Shader uniform type: vec4 (4 float)
    SHADER_UNIFORM_INT,             // Shader uniform type: int
    SHADER_UNIFORM_IVEC2,           // Shader uniform type: ivec2 (2 int)
    SHADER_UNIFORM_IVEC3,           // Shader uniform type: ivec3 (3 int)
    SHADER_UNIFORM_IVEC4,           // Shader uniform type: ivec4 (4 int)
    SHADER_UNIFORM_SAMPLER2D        // Shader uniform type: sampler2d
} ShaderUniformDataType;

// Shader attribute data types
typedef enum {
    SHADER_ATTRIB_FLOAT = 0,        // Shader attribute type: float
    SHADER_ATTRIB_VEC2,             // Shader attribute type: vec2 (2 float)
    SHADER_ATTRIB_VEC3,             // Shader attribute type: vec3 (3 float)
    SHADER_ATTRIB_VEC4              // Shader attribute type: vec4 (4 float)
} ShaderAttributeDataType;

// Pixel formats
// NOTE: Support depends on OpenGL version and platform
typedef enum {
    PIXELFORMAT_UNCOMPRESSED_GRAYSCALE = 1, // 8 bit per pixel (no alpha)
    PIXELFORMAT_UNCOMPRESSED_GRAY_ALPHA,    // 8*2 bpp (2 channels)
    PIXELFORMAT_UNCOMPRESSED_R5G6B5,        // 16 bpp
    PIXELFORMAT_UNCOMPRESSED_R8G8B8,        // 24 bpp
    PIXELFORMAT_UNCOMPRESSED_R5G5B5A1,      // 16 bpp (1 bit alpha)
    PIXELFORMAT_UNCOMPRESSED_R4G4B4A4,      // 16 bpp (4 bit alpha)
    PIXELFORMAT_UNCOMPRESSED_R8G8B8A8,      // 32 bpp
    PIXELFORMAT_UNCOMPRESSED_R32,           // 32 bpp (1 channel - float)
    PIXELFORMAT_UNCOMPRESSED_R32G32B32,     // 32*3 bpp (3 channels - float)
    PIXELFORMAT_UNCOMPRESSED_R32G32B32A32,  // 32*4 bpp (4 channels - float)
    PIXELFORMAT_UNCOMPRESSED_R16,           // 16 bpp (1 channel - half float)
    PIXELFORMAT_UNCOMPRESSED_R16G16B16,     // 16*3 bpp (3 channels - half float)
    PIXELFORMAT_UNCOMPRESSED_R16G16B16A16,  // 16*4 bpp (4 channels - half float)
    PIXELFORMAT_COMPRESSED_DXT1_RGB,        // 4 bpp (no alpha)
    PIXELFORMAT_COMPRESSED_DXT1_RGBA,       // 4 bpp (1 bit alpha)
    PIXELFORMAT_COMPRESSED_DXT3_RGBA,       // 8 bpp
    PIXELFORMAT_COMPRESSED_DXT5_RGBA,       // 8 bpp
    PIXELFORMAT_COMPRESSED_ETC1_RGB,        // 4 bpp
    PIXELFORMAT_COMPRESSED_ETC2_RGB,        // 4 bpp
    PIXELFORMAT_COMPRESSED_ETC2_EAC_RGBA,   // 8 bpp
    PIXELFORMAT_COMPRESSED_PVRT_RGB,        // 4 bpp
    PIXELFORMAT_COMPRESSED_PVRT_RGBA,       // 4 bpp
    PIXELFORMAT_COMPRESSED_ASTC_4x4_RGBA,   // 8 bpp
    PIXELFORMAT_COMPRESSED_ASTC_8x8_RGBA    // 2 bpp
} PixelFormat;

// Texture parameters: filter mode
// NOTE 1: Filtering considers mipmaps if available in the texture
// NOTE 2: Filter is accordingly set for minification and magnification
typedef enum {
    TEXTURE_FILTER_POINT = 0,               // No filter, just pixel approximation
    TEXTURE_FILTER_BILINEAR,                // Linear filtering
    TEXTURE_FILTER_TRILINEAR,               // Trilinear filtering (linear with
mipmaps)
    TEXTURE_FILTER_ANISOTROPIC_4X,          // Anisotropic filtering 4x
    TEXTURE_FILTER_ANISOTROPIC_8X,          // Anisotropic filtering 8x
    TEXTURE_FILTER_ANISOTROPIC_16X,         // Anisotropic filtering 16x
} TextureFilter;

// Texture parameters: wrap mode
typedef enum {
    TEXTURE_WRAP_REPEAT = 0,                // Repeats texture in tiled mode
    TEXTURE_WRAP_CLAMP,                     // Clamps texture to edge pixel in tiled
mode
    TEXTURE_WRAP_MIRROR_REPEAT,             // Mirrors and repeats the texture in
tiled mode
    TEXTURE_WRAP_MIRROR_CLAMP               // Mirrors and clamps to border the
texture in tiled mode
} TextureWrap;

// Cubemap layouts
typedef enum {
    CUBEMAP_LAYOUT_AUTO_DETECT = 0,         // Automatically detect layout type
    CUBEMAP_LAYOUT_LINE_VERTICAL,           // Layout is defined by a vertical line
with faces
    CUBEMAP_LAYOUT_LINE_HORIZONTAL,         // Layout is defined by a horizontal line
with faces
    CUBEMAP_LAYOUT_CROSS_THREE_BY_FOUR,     // Layout is defined by a 3x4 cross with
cubemap faces
    CUBEMAP_LAYOUT_CROSS_FOUR_BY_THREE     // Layout is defined by a 4x3 cross with
cubemap faces
} CubemapLayout;

// Font type, defines generation method
typedef enum {
    FONT_DEFAULT = 0,               // Default font generation, anti-aliased
    FONT_BITMAP,                    // Bitmap font generation, no anti-aliasing
    FONT_SDF                        // SDF font generation, requires external shader
} FontType;

```

```
// Color blending modes (pre-defined)
typedef enum {
    BLEND_ALPHA = 0,                // Blend textures considering alpha (default)
    BLEND_ADDITIVE,                 // Blend textures adding colors
    BLEND_MULTIPLIED,               // Blend textures multiplying colors
    BLEND_ADD_COLORS,               // Blend textures adding colors (alternative)
    BLEND_SUBTRACT_COLORS,          // Blend textures subtracting colors (alternative)
    BLEND_ALPHA_PREMULTIPLY,        // Blend premultiplied textures considering alpha
    BLEND_CUSTOM,                   // Blend textures using custom src/dst factors (use rlSetBlendFactors())
    BLEND_CUSTOM_SEPARATE           // Blend textures using custom rgb/alpha separate src/dst factors (use rlSetBlendFactorsSeparate())
} BlendMode;

// Gesture
// NOTE: Provided as bit-wise flags to enable only desired gestures
typedef enum {
    GESTURE_NONE        = 0,        // No gesture
    GESTURE_TAP         = 1,        // Tap gesture
    GESTURE_DOUBLETAP   = 2,        // Double tap gesture
    GESTURE_HOLD        = 4,        // Hold gesture
    GESTURE_DRAG        = 8,        // Drag gesture
    GESTURE_SWIPE_RIGHT = 16,       // Swipe right gesture
    GESTURE_SWIPE_LEFT  = 32,       // Swipe left gesture
    GESTURE_SWIPE_UP    = 64,       // Swipe up gesture
    GESTURE_SWIPE_DOWN  = 128,      // Swipe down gesture
    GESTURE_PINCH_IN    = 256,      // Pinch in gesture
    GESTURE_PINCH_OUT   = 512       // Pinch out gesture
} Gesture;

// Camera system modes
typedef enum {
    CAMERA_CUSTOM = 0,              // Camera custom, controlled by user (UpdateCamera() does nothing)
    CAMERA_FREE,                    // Camera free mode
    CAMERA_ORBITAL,                 // Camera orbital, around target, zoom supported
    CAMERA_FIRST_PERSON,            // Camera first person
    CAMERA_THIRD_PERSON             // Camera third person
} CameraMode;

// Camera projection
typedef enum {
    CAMERA_PERSPECTIVE = 0,         // Perspective projection
    CAMERA_ORTHOGRAPHIC             // Orthographic projection
} CameraProjection;

// N-patch layout
typedef enum {
    NPATCH_NINE_PATCH = 0,          // Npatch layout: 3x3 tiles
    NPATCH_THREE_PATCH_VERTICAL,    // Npatch layout: 1x3 tiles
    NPATCH_THREE_PATCH_HORIZONTAL   // Npatch layout: 3x1 tiles
} NPatchLayout;

// Callbacks to hook some internal functions
// WARNING: These callbacks are intended for advanced users
typedef void (*TraceLogCallback)(int logLevel, const char *text, va_list args);  // Logging: Redirect trace log messages
typedef unsigned char *(*LoadFileDataCallback)(const char *fileName, int *dataSize); // FileIO: Load binary data
typedef bool (*SaveFileDataCallback)(const char *fileName, void *data, int dataSize); // FileIO: Save binary data
typedef char *(*LoadFileTextCallback)(const char *fileName);            // FileIO: Load text data
typedef bool (*SaveFileTextCallback)(const char *fileName, char *text); // FileIO: Save text data


//------------------------------------------------------------------------------------
// Global Variables Definition
//------------------------------------------------------------------------------------
// It's lonely here...


//------------------------------------------------------------------------------------
```

```
// Window and Graphics Device Functions (Module: core)
//------------------------------------------------------------------------------------

#if defined(__cplusplus)
extern "C" {            // Prevents name mangling of functions
#endif

// Window-related functions
RLAPI void InitWindow(int width, int height, const char *title);  // Initialize window and OpenGL context
RLAPI void CloseWindow(void);                                     // Close window and unload OpenGL context
RLAPI bool WindowShouldClose(void);                               // Check if application should close (KEY_ESCAPE pressed or windows close icon clicked)
RLAPI bool IsWindowReady(void);                                   // Check if window has been initialized successfully
RLAPI bool IsWindowFullscreen(void);                              // Check if window is currently fullscreen
RLAPI bool IsWindowHidden(void);                                  // Check if window is currently hidden
RLAPI bool IsWindowMinimized(void);                               // Check if window is currently minimized
RLAPI bool IsWindowMaximized(void);                               // Check if window is currently maximized
RLAPI bool IsWindowFocused(void);                                 // Check if window is currently focused
RLAPI bool IsWindowResized(void);                                 // Check if window has been resized last frame
RLAPI bool IsWindowState(unsigned int flag);                      // Check if one specific window flag is enabled
RLAPI void SetWindowState(unsigned int flags);                    // Set window configuration state using flags
RLAPI void ClearWindowState(unsigned int flags);                  // Clear window configuration state flags
RLAPI void ToggleFullscreen(void);                                // Toggle window state: fullscreen/windowed, resizes monitor to match window resolution
RLAPI void ToggleBorderlessWindowed(void);                        // Toggle window state: borderless windowed, resizes window to match monitor resolution
RLAPI void MaximizeWindow(void);                                  // Set window state: maximized, if resizable
RLAPI void MinimizeWindow(void);                                  // Set window state: minimized, if resizable
RLAPI void RestoreWindow(void);                                   // Set window state: not minimized/maximized
RLAPI void SetWindowIcon(Image image);                            // Set icon for window (single image, RGBA 32bit)
RLAPI void SetWindowIcons(Image *images, int count);              // Set icon for window (multiple images, RGBA 32bit)
RLAPI void SetWindowTitle(const char *title);                     // Set title for window
RLAPI void SetWindowPosition(int x, int y);                       // Set window position on screen
RLAPI void SetWindowMonitor(int monitor);                         // Set monitor for the current window
RLAPI void SetWindowMinSize(int width, int height);               // Set window minimum dimensions (for FLAG_WINDOW_RESIZABLE)
RLAPI void SetWindowMaxSize(int width, int height);               // Set window maximum dimensions (for FLAG_WINDOW_RESIZABLE)
RLAPI void SetWindowSize(int width, int height);                  // Set window dimensions
RLAPI void SetWindowOpacity(float opacity);                       // Set window opacity [0.0f..1.0f]
RLAPI void SetWindowFocused(void);                                // Set window focused
RLAPI void *GetWindowHandle(void);                                // Get native window handle
RLAPI int GetScreenWidth(void);                                   // Get current screen width
RLAPI int GetScreenHeight(void);                                  // Get current screen height
RLAPI int GetRenderWidth(void);                                   // Get current render width (it considers HiDPI)
RLAPI int GetRenderHeight(void);                                  // Get current render height (it considers HiDPI)
RLAPI int GetMonitorCount(void);                                  // Get number of connected monitors
```

```c
RLAPI int GetCurrentMonitor(void);                                  // Get current monitor where window is placed
RLAPI Vector2 GetMonitorPosition(int monitor);                      // Get specified monitor position
RLAPI int GetMonitorWidth(int monitor);                             // Get specified monitor width (current video mode used by monitor)
RLAPI int GetMonitorHeight(int monitor);                            // Get specified monitor height (current video mode used by monitor)
RLAPI int GetMonitorPhysicalWidth(int monitor);                     // Get specified monitor physical width in millimetres
RLAPI int GetMonitorPhysicalHeight(int monitor);                    // Get specified monitor physical height in millimetres
RLAPI int GetMonitorRefreshRate(int monitor);                       // Get specified monitor refresh rate
RLAPI Vector2 GetWindowPosition(void);                              // Get window position XY on monitor
RLAPI Vector2 GetWindowScaleDPI(void);                              // Get window scale DPI factor
RLAPI const char *GetMonitorName(int monitor);                      // Get the human-readable, UTF-8 encoded name of the specified monitor
RLAPI void SetClipboardText(const char *text);                      // Set clipboard text content
RLAPI const char *GetClipboardText(void);                           // Get clipboard text content
RLAPI Image GetClipboardImage(void);                                // Get clipboard image content
RLAPI void EnableEventWaiting(void);                                // Enable waiting for events on EndDrawing(), no automatic event polling
RLAPI void DisableEventWaiting(void);                               // Disable waiting for events on EndDrawing(), automatic events polling

// Cursor-related functions
RLAPI void ShowCursor(void);                                        // Shows cursor
RLAPI void HideCursor(void);                                        // Hides cursor
RLAPI bool IsCursorHidden(void);                                    // Check if cursor is not visible
RLAPI void EnableCursor(void);                                      // Enables cursor (unlock cursor)
RLAPI void DisableCursor(void);                                     // Disables cursor (lock cursor)
RLAPI bool IsCursorOnScreen(void);                                  // Check if cursor is on the screen

// Drawing-related functions
RLAPI void ClearBackground(Color color);                            // Set background color (framebuffer clear color)
RLAPI void BeginDrawing(void);                                      // Setup canvas (framebuffer) to start drawing
RLAPI void EndDrawing(void);                                        // End canvas drawing and swap buffers (double buffering)
RLAPI void BeginMode2D(Camera2D camera);                            // Begin 2D mode with custom camera (2D)
RLAPI void EndMode2D(void);                                         // Ends 2D mode with custom camera
RLAPI void BeginMode3D(Camera3D camera);                            // Begin 3D mode with custom camera (3D)
RLAPI void EndMode3D(void);                                         // Ends 3D mode and returns to default 2D orthographic mode
RLAPI void BeginTextureMode(RenderTexture2D target);                // Begin drawing to render texture
RLAPI void EndTextureMode(void);                                    // Ends drawing to render texture
RLAPI void BeginShaderMode(Shader shader);                          // Begin custom shader drawing
RLAPI void EndShaderMode(void);                                     // End custom shader drawing (use default shader)
RLAPI void BeginBlendMode(int mode);                                // Begin blending mode (alpha, additive, multiplied, subtract, custom)
RLAPI void EndBlendMode(void);                                      // End blending mode (reset to default: alpha blending)
RLAPI void BeginScissorMode(int x, int y, int width, int height); // Begin scissor mode (define screen area for following drawing)
RLAPI void EndScissorMode(void);                                    // End scissor mode
RLAPI void BeginVrStereoMode(VrStereoConfig config);                // Begin stereo rendering (requires VR simulator)
RLAPI void EndVrStereoMode(void);                                 // End stereo rendering (requires VR simulator)

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

// Shader management functions
// NOTE: Shader functionality is not available on OpenGL 1.1
RLAPI Shader LoadShader(const char *vsFileName, const char *fsFileName);   // Load shader from files and bind default locations
RLAPI Shader LoadShaderFromMemory(const char *vsCode, const char *fsCode); // Load shader from code strings and bind default locations
RLAPI bool IsShaderValid(Shader shader);                                   // Check if a shader is valid (loaded on GPU)
RLAPI int GetShaderLocation(Shader shader, const char *uniformName);       // Get shader uniform location
RLAPI int GetShaderLocationAttrib(Shader shader, const char *attribName);  // Get shader attribute location
RLAPI void SetShaderValue(Shader shader, int locIndex, const void *value, int uniformType);               // Set shader uniform value
RLAPI void SetShaderValueV(Shader shader, int locIndex, const void *value, int uniformType, int count);   // Set shader uniform value vector
RLAPI void SetShaderValueMatrix(Shader shader, int locIndex, Matrix mat);         // Set shader uniform value (matrix 4x4)
RLAPI void SetShaderValueTexture(Shader shader, int locIndex, Texture2D texture); // Set shader uniform value for texture (sampler2d)
RLAPI void UnloadShader(Shader shader);                                    // Unload shader from GPU memory (VRAM)

// Screen-space-related functions
#define GetMouseRay GetScreenToWorldRay     // Compatibility hack for previous raylib versions
RLAPI Ray GetScreenToWorldRay(Vector2 position, Camera camera);         // Get a ray trace from screen position (i.e mouse)
RLAPI Ray GetScreenToWorldRayEx(Vector2 position, Camera camera, int width, int height); // Get a ray trace from screen position (i.e mouse) in a viewport
RLAPI Vector2 GetWorldToScreen(Vector3 position, Camera camera);        // Get the screen space position for a 3d world space position
RLAPI Vector2 GetWorldToScreenEx(Vector3 position, Camera camera, int width, int height); // Get size position for a 3d world space position
RLAPI Vector2 GetWorldToScreen2D(Vector2 position, Camera2D camera);    // Get the screen space position for a 2d camera world space position
RLAPI Vector2 GetScreenToWorld2D(Vector2 position, Camera2D camera);    // Get the world space position for a 2d camera screen space position
RLAPI Matrix GetCameraMatrix(Camera camera);                            // Get camera transform matrix (view matrix)
RLAPI Matrix GetCameraMatrix2D(Camera2D camera);                        // Get camera 2d transform matrix

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

// Custom frame control functions
// NOTE: Those functions are intended for advanced users that want full control over the frame processing
// By default EndDrawing() does this job: draws everything + SwapScreenBuffer() + manage frame timing + PollInputEvents()
// To avoid that behaviour and control frame processes manually, enable in config.h: SUPPORT_CUSTOM_FRAME_CONTROL
RLAPI void SwapScreenBuffer(void);                                // Swap back buffer with front buffer (screen drawing)
RLAPI void PollInputEvents(void);                                 // Register all input events
RLAPI void WaitTime(double seconds);                              // Wait for some time (halt program execution)

// Random values generation functions
```

```c
RLAPI void SetRandomSeed(unsigned int seed);                      // Set the seed for the random number generator
RLAPI int GetRandomValue(int min, int max);                       // Get a random value between min and max (both included)
RLAPI int *LoadRandomSequence(unsigned int count, int min, int max); // Load random values sequence, no values repeated
RLAPI void UnloadRandomSequence(int *sequence);                   // Unload random values sequence

// Misc. functions
RLAPI void TakeScreenshot(const char *fileName);                  // Takes a screenshot of current screen (filename extension defines format)
RLAPI void SetConfigFlags(unsigned int flags);                    // Setup init configuration flags (view FLAGS)
RLAPI void OpenURL(const char *url);                              // Open URL with default system browser (if available)

// NOTE: Following functions implemented in module [utils]
//------------------------------------------------------------------
RLAPI void TraceLog(int logLevel, const char *text, ...);         // Show trace log messages (LOG_DEBUG, LOG_INFO, LOG_WARNING, LOG_ERROR...)
RLAPI void SetTraceLogLevel(int logLevel);                        // Set the current threshold (minimum) log level
RLAPI void *MemAlloc(unsigned int size);                          // Internal memory allocator
RLAPI void *MemRealloc(void *ptr, unsigned int size);             // Internal memory reallocator
RLAPI void MemFree(void *ptr);                                    // Internal memory free

// Set custom callbacks
// WARNING: Callbacks setup is intended for advanced users
RLAPI void SetTraceLogCallback(TraceLogCallback callback);         // Set custom trace log
RLAPI void SetLoadFileDataCallback(LoadFileDataCallback callback); // Set custom file binary data loader
RLAPI void SetSaveFileDataCallback(SaveFileDataCallback callback); // Set custom file binary data saver
RLAPI void SetLoadFileTextCallback(LoadFileTextCallback callback); // Set custom file text data loader
RLAPI void SetSaveFileTextCallback(SaveFileTextCallback callback); // Set custom file text data saver

// Files management functions
RLAPI unsigned char *LoadFileData(const char *fileName, int *dataSize); // Load file data as byte array (read)
RLAPI void UnloadFileData(unsigned char *data);                   // Unload file data allocated by LoadFileData()
RLAPI bool SaveFileData(const char *fileName, void *data, int dataSize); // Save data to file from byte array (write), returns true on success
RLAPI bool ExportDataAsCode(const unsigned char *data, int dataSize, const char *fileName); // Export data to code (.h), returns true on success
RLAPI char *LoadFileText(const char *fileName);                   // Load text data from file (read), returns a '\0' terminated string
RLAPI void UnloadFileText(char *text);                            // Unload file text data allocated by LoadFileText()
RLAPI bool SaveFileText(const char *fileName, char *text);        // Save text data to file (write), string must be '\0' terminated, returns true on success
//------------------------------------------------------------------

// File system functions
RLAPI bool FileExists(const char *fileName);                      // Check if file exists
RLAPI bool DirectoryExists(const char *dirPath);                  // Check if a directory path exists
RLAPI bool IsFileExtension(const char *fileName, const char *ext); // Check file extension (including point: .png, .wav)
RLAPI int GetFileLength(const char *fileName);                    // Get file length in bytes (NOTE: GetFileSize() conflicts with windows.h)
RLAPI const char *GetFileExtension(const char *fileName);         // Get pointer to extension for a filename string (includes dot: '.png')
RLAPI const char *GetFileName(const char *filePath);              // Get pointer to filename for a path string
RLAPI const char *GetFileNameWithoutExt(const char *filePath);    // Get filename string without extension (uses static string)
RLAPI const char *GetDirectoryPath(const char *filePath);         // Get full path for a given fileName with path (uses static string)
RLAPI const char *GetPrevDirectoryPath(const char *dirPath);      // Get previous directory path for a given path (uses static string)
RLAPI const char *GetWorkingDirectory(void);                      // Get current working directory (uses static string)
RLAPI const char *GetApplicationDirectory(void);                  // Get the directory of the running application (uses static string)
RLAPI int MakeDirectory(const char *dirPath);                     // Create directories (including full path requested), returns 0 on success
RLAPI bool ChangeDirectory(const char *dir);                      // Change working directory, return true on success
RLAPI bool IsPathFile(const char *path);                          // Check if a given path is a file or a directory
RLAPI bool IsFileNameValid(const char *fileName);                 // Check if fileName is valid for the platform/OS
RLAPI FilePathList LoadDirectoryFiles(const char *dirPath);       // Load directory filepaths
RLAPI FilePathList LoadDirectoryFilesEx(const char *basePath, const char *filter, bool scanSubdirs); // Load directory filepaths with extension filtering and recursive directory scan. Use 'DIR' in the filter string to include directories in the result
RLAPI void UnloadDirectoryFiles(FilePathList files);              // Unload filepaths
RLAPI bool IsFileDropped(void);                                   // Check if a file has been dropped into window
RLAPI FilePathList LoadDroppedFiles(void);                        // Load dropped filepaths
RLAPI void UnloadDroppedFiles(FilePathList files);                // Unload dropped filepaths
RLAPI long GetFileModTime(const char *fileName);                  // Get file modification time (last write time)

// Compression/Encoding functionality
RLAPI unsigned char *CompressData(const unsigned char *data, int dataSize, int *compDataSize);        // Compress data (DEFLATE algorithm), memory must be MemFree()
RLAPI unsigned char *DecompressData(const unsigned char *compData, int compDataSize, int *dataSize);  // Decompress data (DEFLATE algorithm), memory must be MemFree()
RLAPI char *EncodeDataBase64(const unsigned char *data, int dataSize, int *outputSize);               // Encode data to Base64 string, memory must be MemFree()
RLAPI unsigned char *DecodeDataBase64(const unsigned char *data, int *outputSize);                    // Decode Base64 string data, memory must be MemFree()
RLAPI unsigned int ComputeCRC32(unsigned char *data, int dataSize);     // Compute CRC32 hash code
RLAPI unsigned int *ComputeMD5(unsigned char *data, int dataSize);      // Compute MD5 hash code, returns static int[4] (16 bytes)
RLAPI unsigned int *ComputeSHA1(unsigned char *data, int dataSize);      // Compute SHA1 hash code, returns static int[5] (20 bytes)


// Automation events functionality
RLAPI AutomationEventList LoadAutomationEventList(const char *fileName);                // Load automation events list from file, NULL for empty list, capacity = MAX_AUTOMATION_EVENTS
RLAPI void UnloadAutomationEventList(AutomationEventList list);                         // Unload automation events list from file
RLAPI bool ExportAutomationEventList(AutomationEventList list, const char *fileName);   // Export automation events list as text file
RLAPI void SetAutomationEventList(AutomationEventList *list);                           // Set automation event list to record to
RLAPI void SetAutomationEventBaseFrame(int frame);                                      // Set automation event internal base frame to start recording
RLAPI void StartAutomationEventRecording(void);                                         // Start recording automation events (AutomationEventList must be set)
RLAPI void StopAutomationEventRecording(void);                                          // Stop recording automation events
RLAPI void PlayAutomationEvent(AutomationEvent event);                                  // Play a recorded automation event


//------------------------------------------------------------------------------------
// Input Handling Functions (Module: core)
//------------------------------------------------------------------------------------

// Input-related functions: keyboard
RLAPI bool IsKeyPressed(int key);                             // Check if a key has been pressed once
RLAPI bool IsKeyPressedRepeat(int key);                       // Check if a key has been pressed again
```

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

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

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

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

```

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


//------------------------------------------------------------------------------------
// Camera System Functions (Module: rcamera)
//------------------------------------------------------------------------------------
RLAPI void UpdateCamera(Camera *camera, int mode);      // Update camera position for selected mode
RLAPI void UpdateCameraPro(Camera *camera, Vector3 movement, Vector3 rotation, float zoom); // Update camera movement/rotation


//------------------------------------------------------------------------------------
// Basic Shapes Drawing Functions (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
RLAPI void SetShapesTexture(Texture2D texture, Rectangle source);       // Set texture and rectangle to be used on shapes drawing
RLAPI Texture2D GetShapesTexture(void);                                 // Get texture that is used for shapes drawing
RLAPI Rectangle GetShapesTextureRectangle(void);                        // Get texture source rectangle that is used for shapes drawing

// Basic shapes drawing functions
RLAPI void DrawPixel(int posX, int posY, Color color);                                                   // Draw a pixel using geometry [Can be slow, use with care]
RLAPI void DrawPixelV(Vector2 position, Color color);                                                    // Draw a pixel using geometry (Vector version) [Can be slow, use with care]
RLAPI void DrawLine(int startPosX, int startPosY, int endPosX, int endPosY, Color color);                // Draw a line
RLAPI void DrawLineV(Vector2 startPos, Vector2 endPos, Color color);                                     // Draw a line (using gl lines)
RLAPI void DrawLineEx(Vector2 startPos, Vector2 endPos, float thick, Color color);                       // Draw a line (using triangles/quads)
RLAPI void DrawLineStrip(const Vector2 *points, int pointCount, Color color);                            // Draw lines sequence (using gl lines)
RLAPI void DrawLineBezier(Vector2 startPos, Vector2 endPos, float thick, Color color);                   // Draw line segment cubic-bezier in-out interpolation
RLAPI void DrawCircle(int centerX, int centerY, float radius, Color color);                              // Draw a color-filled circle
RLAPI void DrawCircleSector(Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color);      // Draw a piece of a circle
RLAPI void DrawCircleSectorLines(Vector2 center, float radius, float startAngle, float endAngle, int segments, Color color); // Draw circle sector outline
RLAPI void DrawCircleGradient(int centerX, int centerY, float radius, Color inner, Color outer);         // Draw a gradient-filled circle
RLAPI void DrawCircleV(Vector2 center, float radius, Color color);                                       // Draw a color-filled circle (Vector version)
RLAPI void DrawCircleLines(int centerX, int centerY, float radius, Color color);                         // Draw circle outline
RLAPI void DrawCircleLinesV(Vector2 center, float radius, Color color);                                  // Draw circle outline (Vector version)
RLAPI void DrawEllipse(int centerX, int centerY, float radiusH, float radiusV, Color color);             // Draw ellipse
RLAPI void DrawEllipseLines(int centerX, int centerY, float radiusH, float radiusV, Color color);        // Draw ellipse outline
```

```c
RLAPI void DrawRing(Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color); // Draw ring
RLAPI void DrawRingLines(Vector2 center, float innerRadius, float outerRadius, float startAngle, float endAngle, int segments, Color color);    // Draw ring outline
RLAPI void DrawRectangle(int posX, int posY, int width, int height, Color color);                        // Draw a color-filled rectangle
RLAPI void DrawRectangleV(Vector2 position, Vector2 size, Color color);                                  // Draw a color-filled rectangle (Vector version)
RLAPI void DrawRectangleRec(Rectangle rec, Color color);                                                 // Draw a color-filled rectangle
RLAPI void DrawRectanglePro(Rectangle rec, Vector2 origin, float rotation, Color color);                 // Draw a color-filled rectangle with pro parameters
RLAPI void DrawRectangleGradientV(int posX, int posY, int width, int height, Color top, Color bottom);   // Draw a vertical-gradient-filled rectangle
RLAPI void DrawRectangleGradientH(int posX, int posY, int width, int height, Color left, Color right);   // Draw a horizontal-gradient-filled rectangle
RLAPI void DrawRectangleGradientEx(Rectangle rec, Color topLeft, Color bottomLeft, Color topRight, Color bottomRight); // Draw a gradient-filled rectangle with custom vertex colors
RLAPI void DrawRectangleLines(int posX, int posY, int width, int height, Color color);                   // Draw rectangle outline
RLAPI void DrawRectangleLinesEx(Rectangle rec, float lineThick, Color color);                            // Draw rectangle outline with extended parameters
RLAPI void DrawRectangleRounded(Rectangle rec, float roundness, int segments, Color color);              // Draw rectangle with rounded edges
RLAPI void DrawRectangleRoundedLines(Rectangle rec, float roundness, int segments, Color color);         // Draw rectangle lines with rounded edges
RLAPI void DrawRectangleRoundedLinesEx(Rectangle rec, float roundness, int segments, float lineThick, Color color); // Draw rectangle with rounded edges outline
RLAPI void DrawTriangle(Vector2 v1, Vector2 v2, Vector2 v3, Color color);                                // Draw a color-filled triangle (vertex in counter-clockwise order!)
RLAPI void DrawTriangleLines(Vector2 v1, Vector2 v2, Vector2 v3, Color color);                           // Draw triangle outline (vertex in counter-clockwise order!)
RLAPI void DrawTriangleFan(const Vector2 *points, int pointCount, Color color);                          // Draw a triangle fan defined by points (first vertex is the center)
RLAPI void DrawTriangleStrip(const Vector2 *points, int pointCount, Color color);                        // Draw a triangle strip defined by points
RLAPI void DrawPoly(Vector2 center, int sides, float radius, float rotation, Color color);               // Draw a regular polygon (Vector version)
RLAPI void DrawPolyLines(Vector2 center, int sides, float radius, float rotation, Color color);          // Draw a polygon outline of n sides
RLAPI void DrawPolyLinesEx(Vector2 center, int sides, float radius, float rotation, float lineThick, Color color); // Draw a polygon outline of n sides with extended parameters

// Splines drawing functions
RLAPI void DrawSplineLinear(const Vector2 *points, int pointCount, float thick, Color color);                  // Draw spline: Linear, minimum 2 points
RLAPI void DrawSplineBasis(const Vector2 *points, int pointCount, float thick, Color color);                   // Draw spline: B-Spline, minimum 4 points
RLAPI void DrawSplineCatmullRom(const Vector2 *points, int pointCount, float thick, Color color);               // Draw spline: Catmull-Rom, minimum 4 points
RLAPI void DrawSplineBezierQuadratic(const Vector2 *points, int pointCount, float thick, Color color);          // Draw spline: Quadratic Bezier, minimum 3 points (1 control point): [p1, c2, p3, c4...]
RLAPI void DrawSplineBezierCubic(const Vector2 *points, int pointCount, float thick, Color color);              // Draw spline: Cubic Bezier, minimum 4 points (2 control points): [p1, c2, c3, p4, c5, c6...]
RLAPI void DrawSplineSegmentLinear(Vector2 p1, Vector2 p2, float thick, Color color);                    // Draw spline segment: Linear, 2 points
RLAPI void DrawSplineSegmentBasis(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color); // Draw spline segment: B-Spline, 4 points
RLAPI void DrawSplineSegmentCatmullRom(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float thick, Color color); // Draw spline segment: Catmull-Rom, 4 points
RLAPI void DrawSplineSegmentBezierQuadratic(Vector2 p1, Vector2 c2, Vector2 p3, float thick, Color color); // Draw spline segment: Quadratic Bezier, 2 points, 1 control point
RLAPI void DrawSplineSegmentBezierCubic(Vector2 p1, Vector2 c2, Vector2 c3, Vector2 p4, float thick, Color color); // Draw spline segment: Cubic Bezier, 2 points, 2 control points

// Spline segment point evaluation functions, for a given t [0.0f .. 1.0f]
RLAPI Vector2 GetSplinePointLinear(Vector2 startPos, Vector2 endPos, float t);                           // Get (evaluate) spline point: Linear
RLAPI Vector2 GetSplinePointBasis(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float t);              // Get (evaluate) spline point: B-Spline
RLAPI Vector2 GetSplinePointCatmullRom(Vector2 p1, Vector2 p2, Vector2 p3, Vector2 p4, float t);         // Get (evaluate) spline point: Catmull-Rom
RLAPI Vector2 GetSplinePointBezierQuad(Vector2 p1, Vector2 c2, Vector2 p3, float t);
// Get (evaluate) spline point: Quadratic Bezier
RLAPI Vector2 GetSplinePointBezierCubic(Vector2 p1, Vector2 c2, Vector2 c3, Vector2 p4, float t);        // Get (evaluate) spline point: Cubic Bezier

// Basic shapes collision detection functions
RLAPI bool CheckCollisionRecs(Rectangle rec1, Rectangle rec2);
// Check collision between two rectangles
RLAPI bool CheckCollisionCircles(Vector2 center1, float radius1, Vector2 center2, float radius2);        // Check collision between two circles
RLAPI bool CheckCollisionCircleRec(Vector2 center, float radius, Rectangle rec);
// Check collision between circle and rectangle
RLAPI bool CheckCollisionCircleLine(Vector2 center, float radius, Vector2 p1, Vector2 p2);               // Check if circle collides with a line created betweeen two points [p1] and [p2]
RLAPI bool CheckCollisionPointRec(Vector2 point, Rectangle rec);
// Check if point is inside rectangle
RLAPI bool CheckCollisionPointCircle(Vector2 point, Vector2 center, float radius);
// Check if point is inside circle
RLAPI bool CheckCollisionPointTriangle(Vector2 point, Vector2 p1, Vector2 p2, Vector2 p3);               // Check if point is inside a triangle
RLAPI bool CheckCollisionPointLine(Vector2 point, Vector2 p1, Vector2 p2, int threshold);                // Check if point belongs to line created between two points [p1] and [p2] with defined margin in pixels [threshold]
RLAPI bool CheckCollisionPointPoly(Vector2 point, const Vector2 *points, int pointCount);                // Check if point is within a polygon described by array of vertices
RLAPI bool CheckCollisionLines(Vector2 startPos1, Vector2 endPos1, Vector2 startPos2, Vector2 endPos2, Vector2 *collisionPoint); // Check the collision between two lines defined by two points each, returns collision point by reference
RLAPI Rectangle GetCollisionRec(Rectangle rec1, Rectangle rec2);
// Get collision rectangle for two rectangles collision


//------------------------------------------------------------------------------------
// Texture Loading and Drawing Functions (Module: textures)

//------------------------------------------------------------------------------------

// Image loading functions
// NOTE: These functions do not require GPU access
RLAPI Image LoadImage(const char *fileName);
// Load image from file into CPU memory (RAM)
RLAPI Image LoadImageRaw(const char *fileName, int width, int height, int format, int headerSize);       // Load image from RAW file data
RLAPI Image LoadImageAnim(const char *fileName, int *frames);
// Load image sequence from file (frames appended to image.data)
RLAPI Image LoadImageAnimFromMemory(const char *fileType, const unsigned char *fileData, int dataSize, int *frames); // Load image sequence from memory buffer
RLAPI Image LoadImageFromMemory(const char *fileType, const unsigned char *fileData, int dataSize);      // Load image from memory buffer, fileType refers to extension: i.e. '.png'
RLAPI Image LoadImageFromTexture(Texture2D texture);
// Load image from GPU texture data
RLAPI Image LoadImageFromScreen(void);
// Load image from screen buffer and (screenshot)
RLAPI bool IsImageValid(Image image);
// Check if an image is valid (data and parameters)
RLAPI void UnloadImage(Image image);
// Unload image from CPU memory (RAM)
RLAPI bool ExportImage(Image image, const char *fileName);
// Export image data to file, returns true on success
RLAPI unsigned char *ExportImageToMemory(Image image, const char *fileType, int *fileSize);              // Export image to memory buffer
RLAPI bool ExportImageAsCode(Image image, const char *fileName);
// Export image as code file defining an array of bytes, returns true on success

// Image generation functions
RLAPI Image GenImageColor(int width, int height, Color color);
// Generate image: plain color
RLAPI Image GenImageGradientLinear(int width, int height, int direction, Color start, Color end);        // Generate image: linear gradient, direction in degrees [0..360], 0=Vertical gradient
RLAPI Image GenImageGradientRadial(int width, int height, float density, Color inner, Color outer);      // Generate image: radial gradient
RLAPI Image GenImageGradientSquare(int width, int height, float density, Color inner, Color outer);      // Generate image: square gradient
RLAPI Image GenImageChecked(int width, int height, int checksX, int checksY, Color col1, Color col2);    // Generate image: checked
RLAPI Image GenImageWhiteNoise(int width, int height, float factor); // Generate image: white noise
RLAPI Image GenImagePerlinNoise(int width, int height, int offsetX, int offsetY, float scale);           // Generate image: perlin noise
RLAPI Image GenImageCellular(int width, int height, int tileSize); // Generate image: cellular algorithm, bigger tileSize means bigger cells
RLAPI Image GenImageText(int width, int height, const char *text); // Generate image: grayscale image from text data

// Image manipulation functions
RLAPI Image ImageCopy(Image image); // Create an image duplicate (useful for transformations)
RLAPI Image ImageFromImage(Image image, Rectangle rec); // Create an image from another image piece
RLAPI Image ImageFromChannel(Image image, int selectedChannel); // Create an image from a selected channel of another image (GRAYSCALE)
RLAPI Image ImageText(const char *text, int fontSize, Color color); // Create an image from text (default font)
RLAPI Image ImageTextEx(Font font, const char *text, float fontSize, float spacing, Color tint);          // Create an image from text (custom sprite font)
RLAPI void ImageFormat(Image *image, int newFormat); // Convert image data to desired format
RLAPI void ImageToPOT(Image *image, Color fill); // Convert image to POT (power-of-two)
RLAPI void ImageCrop(Image *image, Rectangle crop); // Crop an image to a defined rectangle
RLAPI void ImageAlphaCrop(Image *image, float threshold); // Crop image depending on alpha value
RLAPI void ImageAlphaClear(Image *image, Color color, float threshold); // Clear alpha channel to desired color
RLAPI void ImageAlphaMask(Image *image, Image alphaMask); // Apply alpha mask to image
RLAPI void ImageAlphaPremultiply(Image *image); // Premultiply alpha channel
RLAPI void ImageBlurGaussian(Image *image, int blurSize); // Apply Gaussian blur using a box blur approximation
RLAPI void ImageKernelConvolution(Image *image, const float *kernel, int kernelSize); // Apply custom square convolution kernel to image
RLAPI void ImageResize(Image *image, int newWidth, int newHeight); // Resize image (Bicubic scaling algorithm)
RLAPI void ImageResizeNN(Image *image, int newWidth,int newHeight); // Resize image (Nearest-Neighbor scaling algorithm)
RLAPI void ImageResizeCanvas(Image *image, int newWidth, int newHeight, int offsetX, int offsetY, Color fill); // Resize canvas and fill with color
RLAPI void ImageMipmaps(Image *image); // Compute all mipmap levels for a provided image
RLAPI void ImageDither(Image *image, int rBpp, int gBpp, int bBpp, int aBpp); // Dither image data to 16bpp or lower (Floyd-Steinberg dithering)
RLAPI void ImageFlipVertical(Image *image); // Flip image vertically
RLAPI void ImageFlipHorizontal(Image *image); // Flip image horizontally
RLAPI void ImageRotate(Image *image, int degrees); // Rotate image by input angle in degrees (-359 to 359)
RLAPI void ImageRotateCW(Image *image); // Rotate image clockwise 90deg
RLAPI void ImageRotateCCW(Image *image); // Rotate image counter-clockwise 90deg
RLAPI void ImageColorTint(Image *image, Color color); // Modify image color: tint
RLAPI void ImageColorInvert(Image *image); // Modify image color: invert
RLAPI void ImageColorGrayscale(Image *image); // Modify image color: grayscale
RLAPI void ImageColorContrast(Image *image, float contrast); // Modify image color: contrast (-100 to 100)
RLAPI void ImageColorBrightness(Image *image, int brightness); // Modify image color: brightness (-255 to 255)
RLAPI void ImageColorReplace(Image *image, Color color, Color replace); // Modify image color: replace color
RLAPI Color *LoadImageColors(Image image); // Load color data from image as a Color array (RGBA - 32bit)
RLAPI Color *LoadImagePalette(Image image, int maxPaletteSize, int *colorCount); // Load colors palette from image as a Color array (RGBA - 32bit)
RLAPI void UnloadImageColors(Color *colors); // Unload color data loaded with LoadImageColors()
RLAPI void UnloadImagePalette(Color *colors); // Unload colors palette loaded with LoadImagePalette()
RLAPI Rectangle GetImageAlphaBorder(Image image, float threshold); // Get image alpha border rectangle
RLAPI 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)
RLAPI void ImageClearBackground(Image *dst, Color color); // Clear image background with given color
RLAPI void ImageDrawPixel(Image *dst, int posX, int posY, Color color); // Draw pixel within an image
RLAPI void ImageDrawPixelV(Image *dst, Vector2 position, Color color); // Draw pixel within an image (Vector version)
RLAPI void ImageDrawLine(Image *dst, int startPosX, int startPosY, int endPosX, int endPosY, Color color); // Draw line within an image
RLAPI void ImageDrawLineV(Image *dst, Vector2 start, Vector2 end, Color color); // Draw line within an image (Vector version)
RLAPI void ImageDrawLineEx(Image *dst, Vector2 start, Vector2 end, int thick, Color color);              // Draw a line defining thickness within an image
RLAPI void ImageDrawCircle(Image *dst, int centerX, int centerY, int radius, Color color);               // Draw a filled circle within an image
RLAPI void ImageDrawCircleV(Image *dst, Vector2 center, int radius, Color color); // Draw a filled circle within an image (Vector version)
RLAPI void ImageDrawCircleLines(Image *dst, int centerX, int centerY, int radius, Color color);          // Draw circle outline within an image
RLAPI void ImageDrawCircleLinesV(Image *dst, Vector2 center, int radius, Color color); // Draw circle outline within an image (Vector version)
RLAPI void ImageDrawRectangle(Image *dst, int posX, int posY, int width, int height, Color color);       // Draw rectangle within an image
RLAPI void ImageDrawRectangleV(Image *dst, Vector2 position, Vector2 size, Color color);                 // Draw rectangle within an image (Vector version)
RLAPI void ImageDrawRectangleRec(Image *dst, Rectangle rec, Color color); // Draw rectangle within an image
RLAPI void ImageDrawRectangleLines(Image *dst, Rectangle rec, int thick, Color color); // Draw rectangle lines within an image
RLAPI void ImageDrawTriangle(Image *dst, Vector2 v1, Vector2 v2, Vector2 v3, Color color);               // Draw triangle within an image
RLAPI void ImageDrawTriangleEx(Image *dst, Vector2 v1, Vector2 v2, Vector2 v3, Color c1, Color c2, Color c3); // Draw triangle with interpolated colors within an image
RLAPI void ImageDrawTriangleLines(Image *dst, Vector2 v1, Vector2 v2, Vector2 v3, Color color);          // Draw triangle outline within an image
RLAPI void ImageDrawTriangleFan(Image *dst, Vector2 *points, int pointCount, Color color);               // Draw a triangle fan defined by points within an image (first vertex is the center)
RLAPI void ImageDrawTriangleStrip(Image *dst, Vector2 *points, int pointCount, Color color);             // Draw a triangle strip defined by points within an image
RLAPI void ImageDraw(Image *dst, Image src, Rectangle srcRec, Rectangle dstRec, Color tint);             // Draw a source image within a destination image (tint applied to source)
RLAPI void ImageDrawText(Image *dst, const char *text, int posX, int posY, int fontSize, Color color);   // Draw text (using default font) within an image (destination)
RLAPI 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
RLAPI Texture2D LoadTexture(const char *fileName); // Load texture from file into GPU memory (VRAM)
RLAPI Texture2D LoadTextureFromImage(Image image); // Load texture from image data
RLAPI TextureCubemap LoadTextureCubemap(Image image, int layout); // Load cubemap from image, multiple image cubemap layouts supported
RLAPI RenderTexture2D LoadRenderTexture(int width, int height); // Load texture for rendering (framebuffer)
RLAPI bool IsTextureValid(Texture2D texture); // Check if a texture is valid (loaded in GPU)
RLAPI void UnloadTexture(Texture2D texture); // Unload texture from GPU memory (VRAM)
RLAPI bool IsRenderTextureValid(RenderTexture2D target); // Check if a render texture is valid (loaded in GPU)
RLAPI void UnloadRenderTexture(RenderTexture2D target); // Unload render texture from GPU memory (VRAM)
RLAPI void UpdateTexture(Texture2D texture, const void *pixels);                                         // Update GPU texture with new data
RLAPI void UpdateTextureRec(Texture2D texture, Rectangle rec, const void *pixels);                       // Update GPU texture rectangle with new data

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

// Texture drawing functions
RLAPI void DrawTexture(Texture2D texture, int posX, int posY, Color tint);                               // Draw a Texture2D
RLAPI void DrawTextureV(Texture2D texture, Vector2 position, Color tint);                                // Draw a Texture2D with position defined as Vector2
RLAPI void DrawTextureEx(Texture2D texture, Vector2 position, float rotation, float scale, Color tint);  // Draw a Texture2D with extended parameters
RLAPI void DrawTextureRec(Texture2D texture, Rectangle source, Vector2 position, Color tint);            // Draw a part of a texture defined by a rectangle
RLAPI void DrawTexturePro(Texture2D texture, Rectangle source, Rectangle dest, Vector2 origin, float rotation, Color tint); // Draw a part of a texture defined by a rectangle with 'pro' parameters
RLAPI void DrawTextureNPatch(Texture2D texture, NPatchInfo nPatchInfo, Rectangle dest, Vector2 origin, float rotation, Color tint); // Draws a texture (or part of it) that stretches or shrinks nicely

// Color/pixel related functions
RLAPI bool ColorIsEqual(Color col1, Color col2);                            // Check if two colors are equal
RLAPI Color Fade(Color color, float alpha);                                 // Get color with alpha applied, alpha goes from 0.0f to 1.0f
RLAPI int ColorToInt(Color color);                                          // Get hexadecimal value for a Color (0xRRGGBBAA)
RLAPI Vector4 ColorNormalize(Color color);                                  // Get Color normalized as float [0..1]
RLAPI Color ColorFromNormalized(Vector4 normalized);                        // Get Color from normalized values [0..1]
RLAPI Vector3 ColorToHSV(Color color);                                      // Get HSV values for a Color, hue [0..360], saturation/value [0..1]
RLAPI Color ColorFromHSV(float hue, float saturation, float value);         // Get a Color from HSV values, hue [0..360], saturation/value [0..1]
RLAPI Color ColorTint(Color color, Color tint);                             // Get color multiplied with another color
RLAPI Color ColorBrightness(Color color, float factor);                     // Get color with brightness correction, brightness factor goes from -1.0f to 1.0f
RLAPI Color ColorContrast(Color color, float contrast);                     // Get color with contrast correction, contrast values between -1.0f and 1.0f
RLAPI Color ColorAlpha(Color color, float alpha);                           // Get color with alpha applied, alpha goes from 0.0f to 1.0f
RLAPI Color ColorAlphaBlend(Color dst, Color src, Color tint);              // Get src alpha-blended into dst color with tint
RLAPI Color ColorLerp(Color color1, Color color2, float factor);            // Get color lerp interpolation between two colors, factor [0.0f..1.0f]
RLAPI Color GetColor(unsigned int hexValue);                                // Get Color structure from hexadecimal value
RLAPI Color GetPixelColor(void *srcPtr, int format);                        // Get Color from a source pixel pointer of certain format
RLAPI void SetPixelColor(void *dstPtr, Color color, int format);            // Set color formatted into destination pixel pointer
RLAPI int GetPixelDataSize(int width, int height, int format);              // Get pixel data size in bytes for certain format


//------------------------------------------------------------------------------------
// Font Loading and Text Drawing Functions (Module: text)
//------------------------------------------------------------------------------------

// Font loading/unloading functions
RLAPI Font GetFontDefault(void);                                                            // Get the default Font
RLAPI Font LoadFont(const char *fileName);                                                  // Load font from file into GPU memory (VRAM)
```

```c
RLAPI Font LoadFontEx(const char *fileName, int fontSize, int *codepoints, int codepointCount); // Load font from file with extended parameters, use NULL for codepoints and 0 for codepointCount to load the default character set, font size is provided in pixels height
RLAPI Font LoadFontFromImage(Image image, Color key, int firstChar); // Load font from Image (XNA style)
RLAPI Font LoadFontFromMemory(const char *fileType, const unsigned char *fileData, int dataSize, int fontSize, int *codepoints, int codepointCount); // Load font from memory buffer, fileType refers to extension: i.e. '.ttf'
RLAPI bool IsFontValid(Font font); // Check if a font is valid (font data loaded, WARNING: GPU texture not checked)
RLAPI GlyphInfo *LoadFontData(const unsigned char *fileData, int dataSize, int fontSize, int *codepoints, int codepointCount, int type); // Load font data for further use
RLAPI Image GenImageFontAtlas(const GlyphInfo *glyphs, Rectangle **glyphRecs, int glyphCount, int fontSize, int padding, int packMethod); // Generate image font atlas using chars info
RLAPI void UnloadFontData(GlyphInfo *glyphs, int glyphCount); // Unload font chars info data (RAM)
RLAPI void UnloadFont(Font font); // Unload font from GPU memory (VRAM)
RLAPI bool ExportFontAsCode(Font font, const char *fileName); // Export font as code file, returns true on success

// Text drawing functions
RLAPI void DrawFPS(int posX, int posY); // Draw current FPS
RLAPI void DrawText(const char *text, int posX, int posY, int fontSize, Color color); // Draw text (using default font)
RLAPI void DrawTextEx(Font font, const char *text, Vector2 position, float fontSize, float spacing, Color tint); // Draw text using font and additional parameters
RLAPI void DrawTextPro(Font font, const char *text, Vector2 position, Vector2 origin, float rotation, float fontSize, float spacing, Color tint); // Draw text using Font and pro parameters (rotation)
RLAPI void DrawTextCodepoint(Font font, int codepoint, Vector2 position, float fontSize, Color tint); // Draw one character (codepoint)
RLAPI void DrawTextCodepoints(Font font, const int *codepoints, int codepointCount, Vector2 position, float fontSize, float spacing, Color tint); // Draw multiple character (codepoint)

// Text font info functions
RLAPI void SetTextLineSpacing(int spacing); // Set vertical line spacing when drawing with line-breaks
RLAPI int MeasureText(const char *text, int fontSize); // Measure string width for default font
RLAPI Vector2 MeasureTextEx(Font font, const char *text, float fontSize, float spacing);    // Measure string size for Font
RLAPI int GetGlyphIndex(Font font, int codepoint); // Get glyph index position in font for a codepoint (unicode character), fallback to '?' if not found
RLAPI GlyphInfo GetGlyphInfo(Font font, int codepoint); // Get glyph font info data for a codepoint (unicode character), fallback to '?' if not found
RLAPI 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)
RLAPI char *LoadUTF8(const int *codepoints, int length);                // Load UTF-8 text encoded from codepoints array
RLAPI void UnloadUTF8(char *text);                                      // Unload UTF-8 text encoded from codepoints array
RLAPI int *LoadCodepoints(const char *text, int *count);                // Load all codepoints from a UTF-8 text string, codepoints count returned by parameter
RLAPI void UnloadCodepoints(int *codepoints);                           // Unload codepoints data from memory
RLAPI int GetCodepointCount(const char *text);                          // Get total number of codepoints in a UTF-8 encoded string
RLAPI int GetCodepoint(const char *text, int *codepointSize);           // Get next codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
RLAPI int GetCodepointNext(const char *text, int *codepointSize);       // Get next codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
RLAPI int GetCodepointPrevious(const char *text, int *codepointSize);   // Get previous codepoint in a UTF-8 encoded string, 0x3f('?') is returned on failure
RLAPI const char *CodepointToUTF8(int codepoint, int *utf8Size);        // Encode one codepoint into UTF-8 byte array (array length returned as parameter)

```

```
// Text strings management functions (no UTF-8 strings, only byte chars)
// NOTE: Some strings allocate memory internally for returned strings, just be careful!
RLAPI int TextCopy(char *dst, const char *src);
// Copy one string to another, returns bytes copied
RLAPI bool TextIsEqual(const char *text1, const char *text2);
// Check if two text string are equal
RLAPI unsigned int TextLength(const char *text);
// Get text length, checks for '\0' ending
RLAPI const char *TextFormat(const char *text, ...);
// Text formatting with variables (sprintf() style)
RLAPI const char *TextSubtext(const char *text, int position, int length);
// Get a piece of a text string
RLAPI char *TextReplace(const char *text, const char *replace, const char *by);
// Replace text string (WARNING: memory must be freed!)
RLAPI char *TextInsert(const char *text, const char *insert, int position);
// Insert text in a position (WARNING: memory must be freed!)
RLAPI const char *TextJoin(const char **textList, int count, const char *delimiter);
// Join text strings with delimiter
RLAPI const char **TextSplit(const char *text, char delimiter, int *count);
// Split text into multiple strings
RLAPI void TextAppend(char *text, const char *append, int *position);
// Append text at specific position and move cursor!
RLAPI int TextFindIndex(const char *text, const char *find);
// Find first text occurrence within a string
RLAPI const char *TextToUpper(const char *text);                      // Get upper
case version of provided string
RLAPI const char *TextToLower(const char *text);                      // Get lower
case version of provided string
RLAPI const char *TextToPascal(const char *text);                    // Get Pascal
case notation version of provided string
RLAPI const char *TextToSnake(const char *text);                      // Get Snake
case notation version of provided string
RLAPI const char *TextToCamel(const char *text);                      // Get Camel
case notation version of provided string

RLAPI int TextToInteger(const char *text);                          // Get integer
value from text (negative values not supported)
RLAPI float TextToFloat(const char *text);                            // Get float
value from text (negative values not supported)


//------------------------------------------------------------------------------------
// Basic 3d Shapes Drawing Functions (Module: models)

//------------------------------------------------------------------------------------

// Basic geometric 3D shapes drawing functions
RLAPI void DrawLine3D(Vector3 startPos, Vector3 endPos, Color color);
// Draw a line in 3D world space
RLAPI void DrawPoint3D(Vector3 position, Color color);
// Draw a point in 3D space, actually a small line
RLAPI void DrawCircle3D(Vector3 center, float radius, Vector3 rotationAxis, float
rotationAngle, Color color); // Draw a circle in 3D world space
RLAPI void DrawTriangle3D(Vector3 v1, Vector3 v2, Vector3 v3, Color color);
// Draw a color-filled triangle (vertex in counter-clockwise order!)
RLAPI void DrawTriangleStrip3D(const Vector3 *points, int pointCount, Color color);
// Draw a triangle strip defined by points
RLAPI void DrawCube(Vector3 position, float width, float height, float length, Color
color);             // Draw cube
RLAPI void DrawCubeV(Vector3 position, Vector3 size, Color color);
// Draw cube (Vector version)
RLAPI void DrawCubeWires(Vector3 position, float width, float height, float length,
Color color);        // Draw cube wires
RLAPI void DrawCubeWiresV(Vector3 position, Vector3 size, Color color);
// Draw cube wires (Vector version)
RLAPI void DrawSphere(Vector3 centerPos, float radius, Color color);
// Draw sphere
RLAPI void DrawSphereEx(Vector3 centerPos, float radius, int rings, int slices, Color
color);             // Draw sphere with extended parameters
RLAPI void DrawSphereWires(Vector3 centerPos, float radius, int rings, int slices,
Color color);         // Draw sphere wires
RLAPI void DrawCylinder(Vector3 position, float radiusTop, float radiusBottom, float
height, int slices, Color color); // Draw a cylinder/cone
RLAPI 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
```

```c
RLAPI void DrawCylinderWires(Vector3 position, float radiusTop, float radiusBottom, float height, int slices, Color color); // Draw a cylinder/cone wires
RLAPI 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
RLAPI void DrawCapsule(Vector3 startPos, Vector3 endPos, float radius, int slices, int rings, Color color); // Draw a capsule with the center of its sphere caps at startPos and endPos
RLAPI void DrawCapsuleWires(Vector3 startPos, Vector3 endPos, float radius, int slices, int rings, Color color); // Draw capsule wireframe with the center of its sphere caps at startPos and endPos
RLAPI void DrawPlane(Vector3 centerPos, Vector2 size, Color color); // Draw a plane XZ
RLAPI void DrawRay(Ray ray, Color color); // Draw a ray line
RLAPI void DrawGrid(int slices, float spacing); // Draw a grid (centered at (0, 0, 0))


//------------------------------------------------------------------------------------------
// Model 3d Loading and Drawing Functions (Module: models)
//------------------------------------------------------------------------------------------

// Model management functions
RLAPI Model LoadModel(const char *fileName); // Load model from files (meshes and materials)
RLAPI Model LoadModelFromMesh(Mesh mesh); // Load model from generated mesh (default material)
RLAPI bool IsModelValid(Model model); // Check if a model is valid (loaded in GPU, VAO/VBOs)
RLAPI void UnloadModel(Model model); // Unload model (including meshes) from memory (RAM and/or VRAM)
RLAPI BoundingBox GetModelBoundingBox(Model model); // Compute model bounding box limits (considers all meshes)

// Model drawing functions
RLAPI void DrawModel(Model model, Vector3 position, float scale, Color tint); // Draw a model (with texture if set)
RLAPI void DrawModelEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model with extended parameters
RLAPI void DrawModelWires(Model model, Vector3 position, float scale, Color tint); // Draw a model wires (with texture if set)
RLAPI 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
RLAPI void DrawModelPoints(Model model, Vector3 position, float scale, Color tint); // Draw a model as points
RLAPI void DrawModelPointsEx(Model model, Vector3 position, Vector3 rotationAxis, float rotationAngle, Vector3 scale, Color tint); // Draw a model as points with extended parameters
RLAPI void DrawBoundingBox(BoundingBox box, Color color); // Draw bounding box (wires)
RLAPI void DrawBillboard(Camera camera, Texture2D texture, Vector3 position, float scale, Color tint);   // Draw a billboard texture
RLAPI void DrawBillboardRec(Camera camera, Texture2D texture, Rectangle source, Vector3 position, Vector2 size, Color tint); // Draw a billboard texture defined by source
RLAPI 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
RLAPI void UploadMesh(Mesh *mesh, bool dynamic); // Upload mesh vertex data in GPU and provide VAO/VBO ids
RLAPI void UpdateMeshBuffer(Mesh mesh, int index, const void *data, int dataSize, int offset); // Update mesh vertex data in GPU for a specific buffer index
RLAPI void UnloadMesh(Mesh mesh); // Unload mesh data from CPU and GPU
RLAPI void DrawMesh(Mesh mesh, Material material, Matrix transform); // Draw a 3d mesh with material and transform
RLAPI void DrawMeshInstanced(Mesh mesh, Material material, const Matrix *transforms, int instances); // Draw multiple mesh instances with material and different transforms
RLAPI BoundingBox GetMeshBoundingBox(Mesh mesh); // Compute mesh bounding box limits
RLAPI void GenMeshTangents(Mesh *mesh); // Compute mesh tangents
RLAPI bool ExportMesh(Mesh mesh, const char *fileName);
// Export mesh data to file, returns true on success
RLAPI bool ExportMeshAsCode(Mesh mesh, const char *fileName);
// Export mesh as code file (.h) defining multiple arrays of vertex attributes

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

// Material loading/unloading functions
RLAPI Material *LoadMaterials(const char *fileName, int *materialCount);
// Load materials from model file
RLAPI Material LoadMaterialDefault(void);
// Load default material (Supports: DIFFUSE, SPECULAR, NORMAL maps)
RLAPI bool IsMaterialValid(Material material);
// Check if a material is valid (shader assigned, map textures loaded in GPU)
RLAPI void UnloadMaterial(Material material);
// Unload material from GPU memory (VRAM)
RLAPI void SetMaterialTexture(Material *material, int mapType, Texture2D texture);
// Set texture for a material map type (MATERIAL_MAP_DIFFUSE, MATERIAL_MAP_SPECULAR...)
RLAPI void SetModelMeshMaterial(Model *model, int meshId, int materialId);
// Set material for a mesh

// Model animations loading/unloading functions
RLAPI ModelAnimation *LoadModelAnimations(const char *fileName, int *animCount);
// Load model animations from file
RLAPI void UpdateModelAnimation(Model model, ModelAnimation anim, int frame);
// Update model animation pose (CPU)
RLAPI void UpdateModelAnimationBones(Model model, ModelAnimation anim, int frame);
// Update model animation mesh bone matrices (GPU skinning)
RLAPI void UnloadModelAnimation(ModelAnimation anim);
// Unload animation data
RLAPI void UnloadModelAnimations(ModelAnimation *animations, int animCount);
// Unload animation array data
RLAPI bool IsModelAnimationValid(Model model, ModelAnimation anim);
// Check model animation skeleton match

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


//------------------------------------------------------------------------------------
```

```
// Audio Loading and Playing Functions (Module: audio)
//------------------------------------------------------------------------------------
typedef void (*AudioCallback)(void *bufferData, unsigned int frames);

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

// Wave/Sound loading/unloading functions
RLAPI Wave LoadWave(const char *fileName);                            // Load wave data from file
RLAPI Wave LoadWaveFromMemory(const char *fileType, const unsigned char *fileData, int dataSize); // Load wave from memory buffer, fileType refers to extension: i.e. '.wav'
RLAPI bool IsWaveValid(Wave wave);                                    // Checks if wave data is valid (data loaded and parameters)
RLAPI Sound LoadSound(const char *fileName);                          // Load sound from file
RLAPI Sound LoadSoundFromWave(Wave wave);                             // Load sound from wave data
RLAPI Sound LoadSoundAlias(Sound source);                             // Create a new sound that shares the same sample data as the source sound, does not own the sound data
RLAPI bool IsSoundValid(Sound sound);                                 // Checks if a sound is valid (data loaded and buffers initialized)
RLAPI void UpdateSound(Sound sound, const void *data, int sampleCount); // Update sound buffer with new data
RLAPI void UnloadWave(Wave wave);                                     // Unload wave data
RLAPI void UnloadSound(Sound sound);                                  // Unload sound
RLAPI void UnloadSoundAlias(Sound alias);                             // Unload a sound alias (does not deallocate sample data)
RLAPI bool ExportWave(Wave wave, const char *fileName);               // Export wave data to file, returns true on success
RLAPI bool ExportWaveAsCode(Wave wave, const char *fileName);         // Export wave sample data to code (.h), returns true on success

// Wave/Sound management functions
RLAPI void PlaySound(Sound sound);                                    // Play a sound
RLAPI void StopSound(Sound sound);                                    // Stop playing a sound
RLAPI void PauseSound(Sound sound);                                   // Pause a sound
RLAPI void ResumeSound(Sound sound);                                  // Resume a paused sound
RLAPI bool IsSoundPlaying(Sound sound);                               // Check if a sound is currently playing
RLAPI void SetSoundVolume(Sound sound, float volume);                 // Set volume for a sound (1.0 is max level)
RLAPI void SetSoundPitch(Sound sound, float pitch);                   // Set pitch for a sound (1.0 is base level)
RLAPI void SetSoundPan(Sound sound, float pan);                       // Set pan for a sound (0.5 is center)
RLAPI Wave WaveCopy(Wave wave);                                       // Copy a wave to a new wave
RLAPI void WaveCrop(Wave *wave, int initFrame, int finalFrame);       // Crop a wave to defined frames range
RLAPI void WaveFormat(Wave *wave, int sampleRate, int sampleSize, int channels); // Convert wave data to desired format
RLAPI float *LoadWaveSamples(Wave wave);                              // Load samples data from wave as a 32bit float data array
RLAPI void UnloadWaveSamples(float *samples);                         // Unload samples data loaded with LoadWaveSamples()

// Music management functions
RLAPI Music LoadMusicStream(const char *fileName);                    // Load music stream from file
```

```c
RLAPI Music LoadMusicStreamFromMemory(const char *fileType, const unsigned char *data, int dataSize); // Load music stream from data
RLAPI bool IsMusicValid(Music music);                                 // Checks if a music stream is valid (context and buffers initialized)
RLAPI void UnloadMusicStream(Music music);                            // Unload music stream
RLAPI void PlayMusicStream(Music music);                              // Start music playing
RLAPI bool IsMusicStreamPlaying(Music music);                         // Check if music is playing
RLAPI void UpdateMusicStream(Music music);                            // Updates buffers for music streaming
RLAPI void StopMusicStream(Music music);                              // Stop music playing
RLAPI void PauseMusicStream(Music music);                             // Pause music playing
RLAPI void ResumeMusicStream(Music music);                            // Resume playing paused music
RLAPI void SeekMusicStream(Music music, float position);              // Seek music to a position (in seconds)
RLAPI void SetMusicVolume(Music music, float volume);                 // Set volume for music (1.0 is max level)
RLAPI void SetMusicPitch(Music music, float pitch);                   // Set pitch for a music (1.0 is base level)
RLAPI void SetMusicPan(Music music, float pan);                       // Set pan for a music (0.5 is center)
RLAPI float GetMusicTimeLength(Music music);                          // Get music time length (in seconds)
RLAPI float GetMusicTimePlayed(Music music);                          // Get current music time played (in seconds)

// AudioStream management functions
RLAPI AudioStream LoadAudioStream(unsigned int sampleRate, unsigned int sampleSize, unsigned int channels); // Load audio stream (to stream raw audio pcm data)
RLAPI bool IsAudioStreamValid(AudioStream stream);                    // Checks if an audio stream is valid (buffers initialized)
RLAPI void UnloadAudioStream(AudioStream stream);                     // Unload audio stream and free memory
RLAPI void UpdateAudioStream(AudioStream stream, const void *data, int frameCount); // Update audio stream buffers with data
RLAPI bool IsAudioStreamProcessed(AudioStream stream);                // Check if any audio stream buffers requires refill
RLAPI void PlayAudioStream(AudioStream stream);                       // Play audio stream
RLAPI void PauseAudioStream(AudioStream stream);                      // Pause audio stream
RLAPI void ResumeAudioStream(AudioStream stream);                     // Resume audio stream
RLAPI bool IsAudioStreamPlaying(AudioStream stream);                  // Check if audio stream is playing
RLAPI void StopAudioStream(AudioStream stream);                       // Stop audio stream
RLAPI void SetAudioStreamVolume(AudioStream stream, float volume);    // Set volume for audio stream (1.0 is max level)
RLAPI void SetAudioStreamPitch(AudioStream stream, float pitch);      // Set pitch for audio stream (1.0 is base level)
RLAPI void SetAudioStreamPan(AudioStream stream, float pan);          // Set pan for audio stream (0.5 is centered)
RLAPI void SetAudioStreamBufferSizeDefault(int size);                 // Default size for new audio streams
RLAPI void SetAudioStreamCallback(AudioStream stream, AudioCallback callback); // Audio thread callback to request new data

RLAPI void AttachAudioStreamProcessor(AudioStream stream, AudioCallback processor); // Attach audio stream processor to stream, receives the samples as 'float'
RLAPI void DetachAudioStreamProcessor(AudioStream stream, AudioCallback processor); // Detach audio stream processor from stream

RLAPI void AttachAudioMixedProcessor(AudioCallback processor); // Attach audio stream processor to the entire audio pipeline, receives the samples as 'float'
RLAPI void DetachAudioMixedProcessor(AudioCallback processor); // Detach audio stream processor from the entire audio pipeline

#if defined(__cplusplus)
}
#endif

```

```c
#endif // RAYLIB_H
```

Sumário

INDEX