Functional Specification

For GNU MAVERIK version 6.1

Jon Cook, Toby Howard

Advanced Interfaces Group
Department of Computer Science
University of Manchester

June 14, 2001

Contents

1 Introduction **1**

1.1 Naming conventions 1

I MAVERIK type specifications **3**

2 Level 1 types **5**

3 Level 2 types **39**

4 Level 3 types **97**

II MAVERIK function specifications **121**

5 Level 1 functions **123**

6 Level 2 functions **233**

7 Level 3 functions **283**

III MAVERIK constant specifications **345**

8 Constants and macros organised by usage **347**

Functions Index **356**

Types, Variables and Constants Index **364**

Chapter 1

Introduction

This document describes the GNU MAVERIK Applications Programmer Interface, and documents constant and type definitions, and function specifications.

The MAVERIK API comprises over 550 functions, only a small subset of which will commonly be used by programmers wishing to use MAVERIK "out of the box". Similarly, many functions will be of interest only to those users wishing to understand the internal workings of MAVERIK, and possibly wishing to tailor it to their own requirements.

With these various requirements in mind, we have divided the MAVERIK functionality into three "levels", which we hope will help users to find their way around.

- **Level 1** functions are those which first-time users of MAVERIK will normally use. These functions make use of the many defaults built into MAVERIK, and should enable users to create MAVERIK applications quickly.
- **Level 2** functions are those which allow more advanced use of MAVERIK. Examples might include defining new classes of object, or defining new methods of navigation.
- **Level 3** functions are intended for "Research and Development" using the MAVERIK system. They are low-level functions which provide interfaces to the MAVERIK kernel and associated modules. For example, Level 3 functions would be required for extending MAVERIK to provide new level-of-detail processing algorithms, new object culling algorithms, or to add kernel support for new kinds of input devices.

Note: This document represents work in-progress. It contains place holders for all of the MAVERIK functions and types, but we simply haven't had the time to fully document, and importantly, cross-link them all. We have concentrated on documenting the most common functions – the others are being steadily added.

1.1 Naming conventions

- All MAVERIK typedefs and constants begin with the prefix `MAV_`
- All MAVERIK functions begin with the prefix `mav_`

- Multi-word function names, such as **mav_frameBegin**, follow the usual conventions of capitalising all but the first word in the function name.

Part I

MAVERIK type specifications

Chapter 2

Level 1 types

MAV_BB

MAVERIK Level 1 typedefs

Summary

Axis-aligned bounding box.

Syntax

```
typedef struct {
  MAV_vector min;
  MAV_vector max;
} MAV_BB;
```

Description

The axis aligned bounding box, `MAV_BB`, comprises two 3D positions, `min` and `max`, to define its extent. It is up to the user to ensure that `max` is greater than `min`.

MAV_SMSObj

MAVERIK Level 1 typedefs

Summary

SMS object list

Syntax

```
typedef struct {
  MAV_SMS *sms;
  MAV_object *selobj;
  MAV_matrix matrix;
  void *userdef;
} MAV_SMSObj;
```

Description

An SMS object contains an SMS, `sms`, of objects which are first transformed by a common transformation matrix, `matrix`, before being transformed by their individual transformation matrix. Objects can be freely added to and removed from the SMS using the usual functions for manipulating SMS's.

An SMS object can be added as an object to any other SMS object, enabling hierarchical structures to be constructed.

If this object is selected via the usual mechanism then `selobj` holds a pointer to the selected subobject.

MAV_box

Summary

Default object class “box”.

Syntax

```
typedef struct {
  MAV_vector size;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_box;
```

Description

An axis-aligned box is defined with its center at the origin. It has a dimension, `size`, along the X,Y and Z axis.

MAV_composite

MAVERIK Level 1 typedefs

Summary

Default object class “composite”.

Syntax

```
typedef struct {
  int numobj;
  MAV_object **obj;
  MAV_BB bb;
  int selobj;
  char *filename;
  MAV_matrix matrix;
  void *userdef;
} MAV_composite;
```

Description

A composite object is a number, `numobj`, of objects which are first transformed by a common transformation matrix, `matrix`, before being transformed by their individual transformation matrices. The objects are defined as an array, `obj`, of pointers to MAVERIK objects. Once defined, the objects comprising the composite objects must remain static, i.e. changing the number of objects in it, or any details of those objects, is forbidden. And since the contents are static, a local coordinate frame bounding box is stored in `bb` for efficiency.

Composite objects are not intended to be defined directly by an application, but rather by functions such as **mav_compositeRead** (page 135), which defines a composite object from a VRML97, Lightwave or AC3D format file.

If a composite object is selected via the usual mechanism then the integer `selobj` holds the array element of the selected sub-object.

MAV_cone

Summary

Default object class “cone”.

Syntax

```
typedef struct {
  float rt;
  float rb;
  float height;
  int nverts;
  int endcap;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_cone;
```

Description

The cone is defined with its centre at the origin and its axis aligned along the Z axis. It has a radius at its top, `rt`, a radius at its bottom, `rb`, and a height, `height`, along the Z axis.

When rendered, `nverts` vertices are used (if greater than two and `mav_opt_curveLOD` is not set) to facet the curved surface of the cone, and the symbolic constant `endcap`, set to `MAV_TRUE` or `MAV_FALSE`, control whether or not the object has endfaces or is effectively hollow.

MAV_ctorus

MAVERIK Level 1 typedefs

Summary

Default object class “circular torus”.

Syntax

```
typedef struct {
  float rmajor;
  float rminor;
  float angle;
  int nverts;
  int nchips;
  int endcap;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_ctorus;
```

Description

The circular torus (a torus with a circular cross section) is defined with its centre at the origin and with a major radius, `rmajor`, a minor radius, `rminor`, and to an angular extent, `angle`, in radians from the X axis around the Z axis.

When rendered, `nverts` vertices are used to facet the curved surface defined by the minor radius, and `nchips` vertices the curved surface defined by the major radius. Both values are only applicable if they are greater than two and `mav_opt_curveLOD` is not set. The symbolic constant `endcap`, set to `MAV_TRUE` or `MAV_FALSE`, control whether or not the object has endfaces or is effectively hollow.

MAV_cylinder

Summary

Default object class “cylinder”.

Syntax

```
typedef struct {
  float radius;
  float height;
  int nverts;
  int endcap;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_cylinder;
```

Description

The cylinder is defined with its centre at the origin and its axis aligned along the Z axis. It has a radius, `radius`, and a height, `height`, along the Z axis.

When rendered, `nverts` vertices are used (if greater than two and `mav_opt_curveLOD` is not set) to facet the curved surface of the cylinder, and the symbolic constant `endcap`, set to `MAV_TRUE` or `MAV_FALSE`, control whether or not the object has endfaces or is effectively hollow.

MAV_ellipse

MAVERIK Level 1 typedefs

Summary

Default object class “ellipse”.

Syntax

```
typedef struct {
  float radius;
  float height;
  int nverts;
  int nchips;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_ellipse;
```

Description

An ellipse is defined with its centre at the origin and with a radius, `height`, along the Z axis and a radius, `radius`, in the XY plane.

When rendered, `nverts` vertices are used to facet the curved surface of the ellipse around the Z axis, and `nchips` vertices around the X axis from -90 to 90 degrees. Both values are only applicable if they are greater than two and `mav_opt_curveLOD` is not set.

MAV_facet

Summary

Default object class “facet”.

Syntax

```
typedef struct {
  int npolys;
  int *np;
  MAV_vector **norm;
  MAV_texCoord **tex;
  MAV_vector **vert;
  MAV_surfaceParams **sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_facet;
```

Description

A facet is a number of polygons which share a common transformation matrix and which allow a normal to be defined for each vertex, rather than for each polygon, thus allowing Gouraud shading across the face of the polygon. They are defined in a similar manner to the polygon group, but with a normal, `norm`, per vertex.

MAV_frameFn

MAVERIK Level 1 typedefs

Summary

The frame function type

Syntax

```
typedef void (*MAV_frameFn)(void *);
```

Description

MAV_hellipse

Summary

Default object class “half ellipse”.

Syntax

```
typedef struct {
  float radius;
  float height;
  int nverts;
  int nchips;
  int endcap;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_hellipse;
```

Description

The half ellipse is defined as the positive Z half-space of an ellipse.

When rendered, `nverts` vertices are used to facet the curved surface of the half ellipse around the Z axis, and `nchips` vertices around the X axis from 0 to 90 degrees. Both values are only applicable if they are greater than two and `mav_opt_curveLOD` is not set. The symbolic constant `endcap`, set to `MAV_TRUE` or `MAV_FALSE`, control whether or not the object has an endface or is effectively hollow.

MAV_hsphere

MAVERIK Level 1 typedefs

Summary

Default object class “half sphere”.

Syntax

```
typedef struct {
  float radius;
  int nverts;
  int nchips;
  int endcap;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_hsphere;
```

Description

The half sphere is defined as the positive Z half-space of a sphere.

When rendered, `nverts` vertices are used to facet the curved surface of the half sphere around the Z axis, and `nchips` vertices around the X axis from 0 to 90 degrees. Both values are only applicable if they are greater than two and `mav_opt_curveLOD` is not set. The symbolic constant `endcap`, set to `MAV_TRUE` or `MAV_FALSE`, control whether or not the object has an endface or is effectively hollow.

MAV_keyboardEvent

Summary

Keyboard event.

Syntax

```
typedef struct {
  MAV_window *win;
  int x;
  int y;
  int root_x;
  int root_y;
  MAV_line line;
  int intersects;
  MAV_object *obj;
  MAV_objectIntersection objint;
  int key;
  int modifiers[MAV_MODIFIER_MAX];
  int movement;
} MAV_keyboardEvent;
```

Description

The keyboard event data structure, `MAV_keyboardEvent`, is passed to the application supplied callback function, set with the **mav_callbackKeyboardSet** (page 132) function, upon keyboard events and details that event.

- `win`
  window in which event occurred.

- `x`, `y`
  position of mouse relative to window origin when event occured.

- `root_x`, `root_y`
  as (`x`, `y`) but relative to root window

- `line`
  the line from the eye point through the world position of the mouse when the keyboard event occured.

- `intersects`
  `MAV_TRUE` if line intersects an object, `MAV_FALSE` otherwise.
- `obj`
  the object intersected by the line (if any).
- `objint`
  the details of any object intersection.
- `key`
  The key pressed. Either this is the ASCII value of the key if printable or a hash defined value for keys such as the function and cursor keys. Note that non-ASCII symbols, such as the pound and euro signs, may not correctly interpreted.
- `modifiers`
  an array containing the status (`MAV_PRESSED` or `MAV_RELEASED`) of the various keyboard modifiers (e.g. `MAV_MODIFIER_SHIFT`, `MAV_MODIFIER_CTRL`, `MAV_MODIFIER_ALT`).
- `movement`
  `MAV_PRESSED` if key down event, `MAV_RELEASED` otherwise

The `MAV_object` which is passed as the first parameter to the event callback function may well be different than `obj` stored in this data structure since the former is always an object of the same type as the callback was registered for. Therefore, event callbacks registered for non-object classes such as `mav_class_any` will receive a dummy object (in this case `mav_object_any`) as their first parameters. No interpretation should be made of these objects by the application. However, the `obj` field always contains the object which the mouse was pointing at when the event occurred regardless of how the callback function was registered.

MAV_line

Summary

Infinite line.

Syntax

```
typedef struct {
  MAV_vector pt;
  MAV_vector dir;
} MAV_line;
```

Description

A `MAV_line` is an infinite line, comprising an origin, `pt`, and a normalised direction vector, `dir`.

MAV_mouseEvent

MAVERIK Level 1 typedefs

Summary

Mouse event.

Syntax

```
typedef struct {
  MAV_window *win;
  int x;
  int y;
  int root_x;
  int root_y;
  MAV_line line;
  int intersects;
  MAV_object *obj;
  MAV_objectIntersection objint;
  int button;
  int modifiers[MAV_MODIFIER_MAX];
  int movement;
} MAV_mouseEvent;
```

Description

The mouse event data structure, `MAV_mouseEvent`, is passed to the application supplied callback function (set with the **mav_callbackMouseSet** (page 133) function) upon mouse button events and details that event.

- `win`
  window in which event occurred.
- `x`, `y`
  position of mouse relative to window origin when event occured.
- `root_x`, `root_y`
  as (x,y) but relative to root window.
- `line`
  the line from the eye point through the world position of the mouse when the mouse event occured.

- `intersects`
  `MAV_TRUE` if line intersects an object, `MAV_FALSE` otherwise.
- `obj`
  the object intersected by the line (if any).
- `objint`
  the details of any object intersection.
- `button`
  `MAV_LEFT_BUTTON`, `MAV_MIDDLE_BUTTON`, `MAV_RIGHT_BUTTON`, `MAV_WHEELUP_BUTTON` or `MAV_WHEELDOWN_BUTTON` to indicate which button generated the event.
- `modifiers`
  an array containing the status (`MAV_PRESSED` or `MAV_RELEASED`) of the various keyboard modifiers (e.g. `MAV_MODIFIER_SHIFT`, `MAV_MODIFIER_CTRL`, `MAV_MODIFIER_ALT`).
- `movement`
  `MAV_PRESSED` if button down event, `MAV_RELEASED` otherwise.

See `MAV_keyboardEvent` for why `obj` will not necessarily be the same as the `MAV_object` passed to the event callback function.

MAV_polygonGrp

MAVERIK Level 1 typedefs

Summary

Default object class “polygon group”.

Syntax

```
typedef struct {
  int npolys;
  int *np;
  MAV_vector *norm;
  MAV_texCoord **tex;
  MAV_vector **vert;
  MAV_surfaceParams **sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_polygonGrp;
```

Description

A polygon group is a number, `npolys`, of polygons, which share a common transformation matrix. Polygon groups can be used to define objects which comprise of many polygons without the rendering inefficiency of each polygon having an individual transformation matrix.

MAV_polygon

Summary

Default object class “polygon”.

Syntax

```
typedef struct {
  int np;
  MAV_vector norm;
  MAV_texCoord *tex;
  MAV_vector *vert;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_polygon;
```

Description

A polygon is defined by a number, `np`, of points, a normal, `norm`, and collection of vertices, `vert`, and, optionally, texture coordinates, `tex`. The polygon must be concave, planar and the vertices ordered anti-clockwise around the normal.

Texture coordinates must be provided is this object is to be textured.

MAV_polyline

MAVERIK Level 1 typedefs

Summary

Default object class “polyline”.

Syntax

```
typedef struct {
  int nlines;
  int *np;
  int *closed;
  MAV_vector **vert;
  MAV_surfaceParams **sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_polyline;
```

Description

A polyline is a number, `nlines`, of lines each consisting of a number, `np`, of vertices, `vert`, each connected by a line. `closed` indicates if the last vertex connects back to the first.

Since it only make sense for this object to be rendered with an emissive solid, attempting to render it with a material or texture gives undefined results.

MAV_pyramid

Summary

Default object class “pyramid”.

Syntax

```
typedef struct {
  float bot_size_x;
  float bot_size_y;
  float top_size_x;
  float top_size_y;
  float offset_x;
  float offset_y;
  float height;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_pyramid;
```

Description

A pyramid is defined with its centre at the origin. Its top and bottom faces, which are in the XY plane, have sizes `top_size_x`, `top_size_y`, `bot_size_x` and `bot_size_y`. The pyramid has a height, `height`, along the Z axis. The X,Y centres of the top and bottom faces are offset by `offset_x` and `offset_y` respectively.

MAV_rectangle

MAVERIK Level 1 typedefs

Summary

Default object class “rectangle”.

Syntax

```
typedef struct {
  float width;
  float height;
  float xtile;
  float ytile;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_rectangle;
```

Description

The rectangle allows for a simple definition of the common case of a 4-vertex polygon centred at the origin with its normal aligned along the positive Z axis. Is is defined by its `width` and `height` along the X and Y axis respectively.

MAV_rtorus

Summary

Default object class “rectangular torus”.

Syntax

```
typedef struct {
  float radius;
  float width;
  float height;
  float angle;
  int nchips;
  int endcap;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_rtorus;
```

Description

The rectangular torus (a torus with a rectangular cross section) is defined with the centre at the origin and with a radius, `radius`, a height, `height`, width, `width` and to an angular extent, `angle`, in radians from the X axis around the Z axis.

When rendered, `nchips` vertices are used (if greater than two and `mav_opt_curveLOD` is not set) to facet the curved surface defined by the radius. The symbolic constant `endcap`, set to `MAV_TRUE` or `MAV_FALSE`, controls whether or not the object has endfaces or is effectively hollow.

MAV_sphere MAVERIK Level 1 typedefs

Summary

Default object class “sphere”.

Syntax

```
typedef struct {
  float radius;
  int nverts;
  int nchips;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_sphere;
```

Description

A sphere is defined with its centre at the origin with a radius `radius`.

When rendered, `nverts` vertices are used to facet the curved surface of the sphere around the Z axis, and `nchips` vertices around the X axis from -90 to 90 degrees. Both values are only applicable if they are greater than two and `mav_opt_curveLOD` is not set.

MAV_surfaceParams

Summary

Surface parameters.

Syntax

```
typedef struct {
  int mode;
  int colour;
  int material;
  int texture;
} MAV_surfaceParams;
```

Description

Every default object contains a field of type `MAV_surfaceParams`, which determines “what colour” is used to render the object. `mode` can take one of the following values:

- `MAV_COLOUR`
  an ambient colour
- `MAV_MATERIAL`
  a material type
- `MAV_TEXTURE`
  a decal texture
- `MAV_LIT_TEXTURE`
  a texture modulated by the material
- `MAV_BLENDED_TEXTURE`
  a blending of the material and texture depending on the texture’s alpha value (0=material, 1=texture).

The other fields, `colour`, `material` and `texture`, respectively specify which colour, material and/or texture index to use from the palette associated with the window in which the object is being drawn. Only for the case of `MAV_LIT_TEXTURE` and `MAV_BLENDED_TEXTURE` does more than one index need to be given.

See also

mav_surfaceParamsNew

MAV_teapot

Summary

Default object class “teapot”.

Syntax

```
typedef struct {
  float size;
  int subdivisions;
  MAV_teabag teabag;
  int lumps;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_teapot;
```

Description

The classic computer graphics teapot without which no Virtual Environment is complete. The teapot is orientated with its Y axis as “up” and the spout pointing along the positive X axis. The teapot has an extent `size` between the edge of the handle and the tip of the spout. The bezier surfaces by which the teapot is defined are subdivided `subdivisions` times when rendered (if greater than zero and `mav_opt_curveLOD` is not set).

The type of tea used to brew-up is governed by the enumerated constant `teabag` which is set to either `TETLEY`, `PG_TIPS` or `EARL_GREY`. The amount of sugar used is controlled by `lumps`, in units of heaped teaspoons, and should be set to less than 2 otherwise you’ll get fat and rot your teeth.

MAV_text

MAVERIK Level 1 typedefs

Summary

Default object class “text”.

Syntax

```
typedef struct {
  char *text;
  int style;
  int justify;
  MAV_surfaceParams *sp;
  MAV_matrix matrix;
  void *userdef;
} MAV_text;
```

Description

The text object allows 3D text to be rendered in a scene. The text, `text`, is defined in the XY plane with the tallest character being approximately 1 unit along the Y axis. The origin is halfway up the text and, depending on the value of `justify`, set to `MAV_LEFT_JUSTIFY`, `MAV_CENTRE_JUSTIFY` or `MAV_RIGHT_JUSTIFY`, is either at the left edge of the text, at its centre or at the right edge respectively. `style` can take the value `MAV_STROKE_FONT`, `MAV_OUTLINE_FONT`, or `MAV_FILLED_FONT`.

MAV_timer

Summary

Timer.

Syntax

```
typedef struct {
  MAV_time start;
  MAV_time end;
  MAV_time elapsed;
  float wall;
  float cpu;
} MAV_timer;
```

Description

The MAVERIK timer data structure is used to give both the wallclock and CPU times that have elapsed between calls to **mav_timerStart** and **mav_timerStop** (page 209).

MAV_vector

MAVERIK Level 1 typedefs

Summary

3D vector.

Syntax

```
typedef struct {
  float x;
  float y;
  float z;
} MAV_vector;
```

Description

A MAVERIK vector comprises 3 floats for the (`x`,`y`,`z`) components. `MAV_vector`'s are used to represent both vectors and coordinate positions.

MAV_viewModifierParams

Summary

View modifier (i.e. stereo) parameters.

Syntax

```
typedef struct {
  float offset;
  float angle;
  void *userdef;
} MAV_viewModifierParams;
```

Description

The `MAV_viewModifierParams` data structure defines the parameters used by the view modifier function of a window to perfrom stereo offset calulations.

The supplied view modifier functions, **mav_eyeLeft** and **mav_eyeRight**, offset the view by +/- `offset`/2.0 along the view right vector. `angle` is currently not used, but could be used by users who want to write view modifier functions which implement a convergence in the view directions. Similarly, `userdef` can be used to expand this data structure further to include any application specific data which could be used in stereo offset calculations.

MAV_viewParams

MAVERIK Level 1 typedefs

Summary

View parameters.

Syntax

```
typedef struct {
  MAV_vector eye;
  MAV_vector view;
  MAV_vector up;
  MAV_vector fixed_up;
  MAV_viewModifierFn mod;
  MAV_vector right;
  MAV_vector trans_eye;
  MAV_vector trans_view;
  MAV_vector trans_up;
  MAV_vector trans_right;
  void *userdef;
} MAV_viewParams;
```

Description

The `MAV_viewParams` data structure defines the user's view, as follows:

- `eye`
  position of the viewer's eye
- `view`
  normalized view direction vector, measured from the eye position
- `up`
  normalized view up vector
- `mod`
  at the start of each frame, the `mod` function is called. This arbitrarily transforms the supplied viewing vectors to produce the viewing vectors `trans_eye`, `trans_view`, `trans_up` and `trans_right`. It is these transformed vectors that are actually used to define the view. Setting this value to NULL, or using the `MAV_viewModifierFn` **mav_viewParamsFixed** (page 214), performs a null transformation (that is, the transformed vectors are the same as the supplied vectors). However, advanced users might wish to write their own functions to implement, say,

an over-the-shoulder view for an avatar or a view whose orientation is matched to that of an HMD.

- `fixed_up`
  normalized world up vector, used by some navigation functions. This field is optional but it is strongly recommended that it is defined.
- `right`
  not directly defined by the user but rather calculated at the start of each frame by the kernel using the other viewing vectors. Since it is used by many routines, it is stored in the data structure rather than being calculated each time it is needed.
- `trans_view`, `trans_up`, `trans_right`
  see description of `mod`.
- `userdef`
  allows arbitrary application data to be added to this structure

The viewing frustum is only fully defined once the `MAV_viewParams` are associated with a window, using the **mav_windowViewParamsSet** (page 232) function, thus defining the field of view, aspect ratio and near/far clip planes.

Chapter 3

Level 2 types

MAV_HBB

MAVERIK Level 2 typedefs

Summary

SMS data structure.

Syntax

```
typedef struct {
  MAV_HBBCluster *root;
  int size;
  MAV_HBBPointer *pointer;
} MAV_HBB;
```

Description

This data structure is supplied for information only.

This is the data structure used to implement the default `MAV_HBB` SMS, namely, a hierarchy of objects (based on bounding boxes).

Users should not manipulate this data structure directly, but rather by the execution of the relevant callback (for example, add object, remove object) which act upon it. Similarly, users will rarely need to explicitly define this data structure since the **mav_HBBNew** (page 329) function return a pointer to a newly created and initialized data structures. This routine act as the second argument to the **mav_SMSNew** function. For example:

```
 sms= mav_SMSNew(mav_SMSClass_HBB, mav_HBBNew());
```

MAV_SMSCallback

Summary

SMS Callback

Syntax

```
typedef struct {
  int num;
} MAV_SMSCallback;
```

Description

MAV_SMSExecFnFn

MAVERIK Level 2 typedefs

Summary

The SMS execute function type

Syntax

```
typedef void (*MAV_SMSExecFnFn)(MAV_object *, MAV_drawInfo *, void *);
```

Description

MAV_SMSExecFn

Summary

SMS Execute function

Syntax

```
typedef struct {
  MAV_SMSExecFnFn fn;
  int nocalc;
  void *params;
} MAV_SMSExecFn;
```

Description

MAV_TDMCursor

MAVERIK Level 2 typedefs

Summary

TDM Cursor

Syntax

```
typedef struct {
  int tracker;
  int style;
  MAV_surfaceParams *sp;
  void *userdef;
} MAV_TDMCursor;
```

Description

MAV_TDMEvent

Summary

TDM Event

Syntax

```
typedef struct {
  MAV_TDMPos pos;
  MAV_line line;
  int intersects;
  MAV_object *obj;
  MAV_objectIntersection objint;
  int button;
  int tracker;
  int movement;
} MAV_TDMEvent;
```

Description

MAV_TDMPos

MAVERIK Level 2 typedefs

Summary

TDM Position

Syntax

```
typedef struct {
  MAV_vector pos;
  MAV_vector u;
  MAV_vector v;
  MAV_vector n;
  MAV_matrix matrix;
  MAV_quaternion quaternion;
} MAV_TDMPos;
```

Description

MAV_avatar

Summary

Avatar

Syntax

```
typedef struct {
  MAV_avatarPart *root;
  MAV_avatarPart *part[19];
  MAV_surfaceParams *sp[5];
  int movement;
  float speed;
  float offset;
  int animFromMat;
  int animate;
  int move;
  MAV_timer timer;
  float time;
  float last_time;
  MAV_vector last_pos;
  MAV_matrix vertical;
  MAV_matrix rotation;
  MAV_matrix matrix;
  MAV_vector right_hand;
  MAV_vector left_hand;
  int holding_right;
  int holding_left;
  MAV_surfaceParams *laser_sp;
  void *userdef;
} MAV_avatar;
```

Description

MAV_callbackBBFn

MAVERIK Level 2 typedefs

Summary

The calculate bounding box callback function type

Syntax

```
typedef int (*MAV_callbackBBFn)(MAV_object *, MAV_BB *);
```

Description

MAV_callbackCrossingFn

Summary

The crossing event callback function type

Syntax

```
typedef int (*MAV_callbackCrossingFn)(MAV_object *, MAV_crossingEvent *);
```

Description

MAV_callbackDeleteFn

MAVERIK Level 2 typedefs

Summary

The delete callback function type

Syntax

```
typedef int (*MAV_callbackDeleteFn)(MAV_object *);
```

Description

MAV_callbackDrawFn

Summary

The draw callback function type

Syntax

```
typedef int (*MAV_callbackDrawFn)(MAV_object *, MAV_drawInfo *);
```

Description

MAV_callbackDumpFn

MAVERIK Level 2 typedefs

Summary

The dump callback function type

Syntax

```
typedef int (*MAV_callbackDumpFn)(MAV_object *);
```

Description

MAV_callbackExposeFn

Summary

The expose event callback function type

Syntax

```
typedef int (*MAV_callbackExposeFn)(MAV_object *, MAV_exposeEvent *);
```

Description

MAV_callbackFn

MAVERIK Level 2 typedefs

Summary

The generic callback function type

Syntax

```
typedef int (*MAV_callbackFn)(MAV_object *, void *, void *);
```

Description

MAV_callbackGetMatrixFn

Summary

The get matrix callback function type

Syntax

```
typedef int (*MAV_callbackGetMatrixFn)(MAV_object *, MAV_matrix **);
```

Description

MAV_callbackGetSurfaceParamsFn

MAVERIK Level 2 typedefs

Summary

The get surface parameters callback function type

Syntax

```
typedef int (*MAV_callbackGetSurfaceParamsFn)(MAV_object *, MAV_surfaceParams ***);
```

Description

MAV_callbackGetUserdefFn

Summary

The get userdef callback function type

Syntax

```
typedef int (*MAV_callbackGetUserdefFn)(MAV_object *, void ***);
```

Description

MAV_callbackIDFn

MAVERIK Level 2 typedefs

Summary

The identify callback function type

Syntax

```
typedef int (*MAV_callbackIDFn)(MAV_object *, char **);
```

Description

MAV_callbackIntersectFn

Summary

The calculate intersection callback function type

Syntax

```
typedef int (*MAV_callbackIntersectFn)(MAV_object *, MAV_line *, MAV_objectIntersection *);
```

Description

MAV_callbackKeyboardFn

MAVERIK Level 2 typedefs

Summary

The keyboard event callback function type

Syntax

```
typedef int (*MAV_callbackKeyboardFn)(MAV_object *, MAV_keyboardEvent *);
```

Description

MAV_callbackMapFn

Summary

The map event callback function type

Syntax

```
typedef int (*MAV_callbackMapFn)(MAV_object *, MAV_mapEvent *);
```

Description

MAV_callbackMouseFn

MAVERIK Level 2 typedefs

Summary

The mouse event callback function type

Syntax

```
typedef int (*MAV_callbackMouseFn)(MAV_object *, MAV_mouseEvent *);
```

Description

MAV_callbackResizeFn

MAVERIK Level 2 typedefs

Summary

The resize event callback function type

Syntax

```
typedef int (*MAV_callbackResizeFn)(MAV_object *, MAV_resizeEvent *);
```

Description

MAV_callbackTDMFn

MAVERIK Level 2 typedefs

Summary

The TDM event callback function type

Syntax

```
typedef int (*MAV_callbackTDMFn)(MAV_object *, MAV_TDMEvent *);
```

Description

MAV_callback

Summary

Callback

Syntax

```
typedef struct {
  int num;
} MAV_callback;
```

Description

MAV_class

MAVERIK Level 2 typedefs

Summary

Maverik class.

Syntax

```
typedef struct {
  MAV_callbackFn fn[MAV_MAX_CBS][MAV_MAX_WIN];
} MAV_class;
```

Description

This data structure is described for information only. Users will never directly access its contents, but it is hoped that by exposing them users will gain an insight into how MAVERIK operates.

A MAVERIK class is simply an array of callback functions – methods which act upon the data structure associated with this class. Note that callback functions are defined on a per class and per window basis, thus allowing for the possibility of different behaviour in different windows. For example, the draw callback for an object may cause it to rendered filled in one window, and rendered in wire-frame in another window.

MAV_clipPlane

Summary

Clip plane.

Syntax

```
typedef struct {
  MAV_vector norm;
  float d;
} MAV_clipPlane;
```

Description

A `MAV_clipPlane` is used to define a clipping plane, and consists of the plane normal, `norm`, and a value, `d`, which satisfies the equation `d = norm.pt`, where `pt` is any point on the plane.

The positive half space of the clipping plane defines which objects are to be removed from consideration.

MAV_clipPlanes

MAVERIK Level 2 typedefs

Summary

Set of clipping planes.

Syntax

```
typedef struct {
  int num;
  MAV_clipPlane planes[MAV_MAX_CLIP_PLANES];
} MAV_clipPlanes;
```

Description

The data structure defines a set of clipping planes, comprising a number, `num`, of clip planes `planes` upto a maximum of `MAV_MAX_CLIP_PLANES` (currently 10).

MAV_colour

Summary

Colour

Syntax

```
typedef struct {
  int id;
  int defwarn;
  int defined;
  float colour[4];
} MAV_colour;
```

Description

MAV_compositeFormat

MAVERIK Level 2 typedefs

Summary

The supported composite file formats data structure

Syntax

```
typedef struct {
  int defined;
  char *ext;
  MAV_compositeReadFn fn;
} MAV_compositeFormat;
```

Description

MAV_compositeReadFn

Summary

The composite file parser function

Syntax

```
typedef int (*MAV_compositeReadFn)(char *, MAV_composite *, MAV_matrix);
```

Description

MAV_crossingEvent

MAVERIK Level 2 typedefs

Summary

Mouse crossing events.

Syntax

```
typedef struct {
  MAV_window *win;
  int dir;
} MAV_crossingEvent;
```

Description

The mouse window crossing (enters/leaves) event data structure, `MAV_crossingEvent`, is passesd to the application supplied callback function (set with the **mav_callbackCrossingSet** (page 261) function) to provide details of this event.

The fields are as follows:

- `win`
  window for which event occured.
- `dir`
  set to `MAV_ENTER` or `MAP_LEAVE`, as appropriate.

MAV_drawInfo

Summary

Drawing information.

Syntax

```
typedef struct {
  MAV_clipPlanes cp;
  MAV_viewParams vp;
  void *userdef;
} MAV_drawInfo;
```

Description

The drawing information data structure, `MAV_drawInfo`, is calculated in **mav_SMSDisplay** (page 127) and subsequently forms part of the information that is passed to rendering callbacks. The information consists of the view frustum's clipping planes `cp`, the viewing parameters `vp`, and, for advanced users, a user definable field `userdef`, to allow for application-specific data to be added to this structure.

The rendering callback for an object can make use of the information in this data structure to perform fine culling or level-of-detail processing, for example.

MAV_exposeEvent

MAVERIK Level 2 typedefs

Summary

Window expose events.

Syntax

```
typedef struct {
  MAV_window *win;
} MAV_exposeEvent;
```

Description

The window expose event data structure, `MAV_exposeEvent`, is passesd to the application supplied callback function (set with the **mav_callbackExposeSet** (page 261) function) to provide details of this event. Its only field is the window, `win`, for which the event occured.

MAV_font

Summary

Font

Syntax

```
typedef struct {
  int id;
  int defwarn;
  int defined;
  char *name;
  int width[256];
} MAV_font;
```

Description

MAV_light

MAVERIK Level 2 typedefs

Summary

Light

Syntax

```
typedef struct {
  int id;
  int index;
  int defwarn;
  int defined;
  float ambient[4];
  float diffuse[4];
  float specular[4];
  MAV_vector pos;
  int positioning;
} MAV_light;
```

Description

MAV_lightingModel

Summary

Lighting Model

Syntax

```
typedef struct {
  int id;
  int defwarn;
  int defined;
  float ambient[4];
  int localviewer;
} MAV_lightingModel;
```

Description

MAV_list

MAVERIK Level 2 typedefs

Summary

Generic Maverik list

Syntax

```
typedef struct MAV_LISTITEM {
  void *data1;
  void *data2;
  struct MAV_LISTITEM *next;
  struct MAV_LISTITEM *prev;
} MAV_listItem;

typedef struct MAV_LISTPOINTER {
  MAV_listItem *item;
  struct MAV_LISTPOINTER *next;
} MAV_listPointer;

typedef struct {
  int length;
  MAV_listItem *head;
  MAV_listItem *tail;
  MAV_listPointer *current;
} MAV_list;
```

Description

MAV_mapEvent

Summary

Window map events.

Syntax

```
typedef struct {
  MAV_window *win;
  int map;
} MAV_mapEvent;
```

Description

The window mapping (iconify/de-iconify) event data structure, `MAV_mapEvent`, is passesd to the application supplied callback function (set with the **mav_callbackMapSet** (page 261) function) to provide details of this event.

The fields are as follows:

- `win`
  window for which event occured.
- `map`
  set to `MAV_MAP` or `MAP_UNMAP`, as appropriate.

MAV_material

MAVERIK Level 2 typedefs

Summary

Material

Syntax

```
typedef struct {
  int id;
  int defwarn;
  int defined;
  float ambient[4];
  float diffuse[4];
  float specular[4];
  float emission[4];
  float shine;
} MAV_material;
```

Description

MAV_matrix

Summary

4x4 transformation matrix.

Syntax

```
typedef struct {
  float mat[4][4];
} MAV_matrix;
```

Description

The MAVERIK 4x4 transformation matrix. It is strongly recommended that matrices should only be manipulated using the functions and hash defines provided (see **mav_matrixSet**, and associated functions). At their own risk, advanced users may access the individual elements stored in `mat`. However, they should be aware of the ordering of the array as discussed in the MAVERIK FAQ and OpenGL documentation.

Note that matrices should not be skewed, nor have a non-uniform scaling applied to them. This is because of assumptions made in the default intersection functions.

MAV_navigatorFn

MAVERIK Level 2 typedefs

Summary

Navigator function

Syntax

```
typedef void (*MAV_navigatorFn)(MAV_viewParams *, float, float, float);
```

Description

MAV_objList

Summary

SMS data structure.

Syntax

```
typedef struct {
  MAV_list *list;
} MAV_objList;
```

Description

This data structure is supplied for information only.

This is the data structure used to implement the default `MAV_objList` SMS, namely, a simple linked list of objects.

Users should not manipulate this data structure directly, but rather by the execution of the relevant callback (for example, add object, remove object) which act upon it. Similarly, users will rarely need to explicitly define this data structure since the **mav_objListNew** (page 329) function return a pointer to a newly created and initialized data structures. This routine act as the second argument to the **mav_SMSNew** function. For example:

```
sms= mav_SMSNew(mav_SMSClass_objList, mav_objListNew());
```

MAV_objectIntersection

MAVERIK Level 2 typedefs

Summary

Object/line intersection.

Syntax

```
typedef struct {
  float pt1;
  float pt2;
  MAV_vector intpt;
  MAV_vector surnorm;
} MAV_objectIntersection;
```

Description

The object intersection data structure, `MAV_objectIntersection`, is used by intersection callback routines to report the details of an object/line intersection test.

Currently, the only data held in this structure is `pt1` – the distance from the line's origin to the first intersection of the line with the object. This value should be set to a negative number if the line does not intersect the object (under these circumstances the intersection callback routine should also return `MAV_FALSE`). If the line originates inside the object, `pt1` is zero.

The `MAV_objectIntersection` for the object with the closest intersection distance forms the `objint` field in event callback data structures such as `MAV_keyboardEvent`.

(This data structure needs re-redesigning, and may well change in the future.)

MAV_object

Summary

Maverik object.

Syntax

```
typedef struct {
  void *the_data;
  MAV_class *the_class;
} MAV_object;
```

Description

A MAVERIK object is simply the encapsulation in a single data structure of a pointer to an object's data, `the_data`, and the methods which act upon it, `the_class`. This gives a generic handle to any object, regardless of class, and is used as the argument to other MAVERIK functions.

Applications would rarely need to explicitly create or directly access the fields of this data structure. This should be achieved by the **mav_objectNew**, **mav_objectDataGet** and **mav_objectClassGet** functions.

See also

MAV_class, **mav_objectNew**, **mav_objectDataGet**, **mav_objectClassGet**

MAV_palette

MAVERIK Level 2 typedefs

Summary

Palette

Syntax

```
typedef struct {
  int defwarn;
  int lm_defwarn;
  MAV_lightingModel lm;
  int light_defwarn;
  MAV_light *lightlist;
  int col_defwarn;
  MAV_colour *collist;
  int mat_defwarn;
  MAV_material *matlist;
  int tex_defwarn;
  MAV_texture *texlist;
  MAV_texEnvFn texEnv;
  int font_defwarn;
  MAV_font *fontlist;
} MAV_palette;
```

Description

MAV_quaternion

Summary

Quaternion.

Syntax

```
typedef struct {
  float w;
  float x;
  float y;
  float z;
} MAV_quaternion;
```

Description

A MAVERIK quaternion comprises 4 floats which represents the quaternion [`w`,(`x`,`y`,`z`)]. As with matrices, it is recommended to manipulate quaternions only with the functions provided (see, for example, **mav_quaternionSet** (page 202)).

MAV_resizeEvent

MAVERIK Level 2 typedefs

Summary

Window resize events.

Syntax

```
typedef struct {
  MAV_window *win;
  int width;
  int height;
} MAV_resizeEvent;
```

Description

The resize event data structure, `MAV_resizeEvent`, is passesd to the application supplied callback function (set with the **mav_callbackResizeSet** (page 261) function) to provide details of this event.

The fields are as follows:

- `win`
  window for which event occured.
- `width`, `height`
  new size of the window.

A default callback routine is registered for this events which updates the window's state with the new size and alters the perspective parameters to account for the new apsect. This callbacks have a return value of -100 which can be checked for by mav_eventsCheck.

MAV_texCoord

Summary

Texture coordinate.

Syntax

```
typedef struct {
  float s;
  float t;
} MAV_texCoord;
```

Description

A MAVERIK texture coordinate comprises two floats, `s` and `t`, representing the texture coordinates at a vertex.

MAV_texEnvFn MAVERIK Level 2 typedefs

Summary

The apply texture environment callback function type

Syntax

```
typedef void (*MAV_texEnvFn)(MAV_texture *);
```

Description

MAV_texture

Summary

Texture

Syntax

```
typedef struct {
  int id;
  int defwarn;
  int defined;
  int width;
  int height;
  unsigned long *mem;
  char *filename;
  MAV_texEnvFn texEnv;
  int transparent;
  int mipmapped;
  int nmaps;
  int *xsize;
  int *ysize;
  unsigned long **mipmap;
} MAV_texture;
```

Description

MAV_time

MAVERIK Level 2 typedefs

Summary

Time.

Syntax

```
typedef struct {
  long sec;
  long usec;
  long cpu;
} MAV_time;
```

Description

The MAVERIK time data structure contains the number of seconds (`sec`) and microseconds (`usec`) since midnight January 1, 1970. Also stored are the number of microseconds of CPU time (`cpu`), since program execution began.

MAV_viewModifierFn

Summary

The view modification function type

Syntax

```
typedef void (*MAV_viewModifierFn)(MAV_window *);
```

Description

MAV_window

MAVERIK Level 2 typedefs

Summary

Window

Syntax

```
typedef struct {
  int id;
  char *name;
  int x;
  int y;
  int width;
  int height;
  MAV_viewParams *vp;
  MAV_viewModifierFn mod;
  MAV_viewModifierParams *vmp;
  MAV_vector eye;
  MAV_vector view;
  MAV_vector up;
  MAV_vector right;
  int orthogonal;
  float ncp;
  float fcp;
  float fov;
  float aspect;
  float offset;
  float angle;
  float ortho_size;
  MAV_matrix viewMat;
  MAV_matrix projMat;
  MAV_matrix pdvMat;
  float background_red;
  float background_green;
  float background_blue;
  MAV_palette *palette;
  MAV_vector ncpv[5];
  MAV_vector fcpv[5];
  void *userdef;
} MAV_window;
```

Description

Chapter 4

Level 3 types

MAV_SMSCallbackDeleteFn

MAVERIK Level 3 typedefs

Summary

The SMS delete callback function type

Syntax

```
typedef int (*MAV_SMSCallbackDeleteFn)(MAV_SMS *, int *);
```

Description

MAV_SMSCallbackEmptyFn

MAVERIK Level 3 typedefs

Summary

The SMS empty callback function type

Syntax

```
typedef int (*MAV_SMSCallbackEmptyFn)(MAV_SMS *, int *);
```

Description

MAV_SMSCallbackExecFnFn

Summary

The SMS execute function callback function type

Syntax

```
typedef int (*MAV_SMSCallbackExecFnFn)(MAV_SMS *, MAV_drawInfo *, MAV_SMSExecFn *);
```

Description

MAV_SMSCallbackFn

MAVERIK Level 3 typedefs

Summary

The generic SMS callback function type

Syntax

```
typedef int (*MAV_SMSCallbackFn)(MAV_SMS *, void *, void *, void *, void *);
```

Description

MAV_SMSCallbackIntersectFn

MAVERIK Level 3 typedefs

Summary

The SMS intersects callback function type

Syntax

```
typedef int (*MAV_SMSCallbackIntersectFn)(MAV_SMS *, MAV_window *, MAV_line *, MAV_objectIntersection *,
```

Description

MAV_SMSCallbackObjectAddFn

MAVERIK Level 3 typedefs

Summary

The SMS add object callback function type

Syntax

```
typedef int (*MAV_SMSCallbackObjectAddFn)(MAV_SMS *, MAV_object *);
```

Description

MAV_SMSCallbackObjectContainsFn

Summary

The SMS contains object callback function type

Syntax

```
typedef int (*MAV_SMSCallbackObjectContainsFn)(MAV_SMS *, MAV_object *, int *);
```

Description

MAV_SMSCallbackObjectNextFn

MAVERIK Level 3 typedefs

Summary

The SMS next object callback function type

Syntax

```
typedef int (*MAV_SMSCallbackObjectNextFn)(MAV_SMS *, MAV_object **);
```

Description

MAV_SMSCallbackObjectRmvFn

Summary

The SMS remove object callback function type

Syntax

```
typedef int (*MAV_SMSCallbackObjectRmvFn)(MAV_SMS *, MAV_object *);
```

Description

MAV_SMSCallbackPointerPopFn

MAVERIK Level 3 typedefs

Summary

The SMS pop pointer callback function type

Syntax

```
typedef int (*MAV_SMSCallbackPointerPopFn)(MAV_SMS *);
```

Description

MAV_SMSCallbackPointerPushFn

Summary

The SMS push pointer callback function type

Syntax

```
typedef int (*MAV_SMSCallbackPointerPushFn)(MAV_SMS *);
```

Description

MAV_SMSCallbackPointerResetFn

MAVERIK Level 3 typedefs

Summary

The SMS reset pointer callback function type

Syntax

```
typedef int (*MAV_SMSCallbackPointerResetFn)(MAV_SMS *);
```

Description

MAV_SMSCallbackSizeFn

Summary

The SMS size callback function type

Syntax

```
typedef int (*MAV_SMSCallbackSizeFn)(MAV_SMS *, int *);
```

Description

MAV_SMSClass

MAVERIK Level 3 typedefs

Summary

Maverik SMS class

Syntax

```
typedef struct {
  MAV_SMSCallbackFn fn[MAV_MAX_CBS];
} MAV_SMSClass;
```

Description

MAV_SMS

Summary

Maverik SMS

Syntax

```
typedef struct {
  void *the_data;
  MAV_SMSClass *the_class;
  int selectable[MAV_MAX_WIN];
  void *userdef;
} MAV_SMS;
```

Description

MAVERIK SMS's are completely analogous to MAVERIK objects in that they are an arbitrary data structure coupled to methods which act upon it.

A MAVERIK SMS is simply the encapsulation in a single data structure of a pointer to an SMS's data, `the_data`, and the methods which act upon it, `the_class`. This gives a generic handle to any SMS, regardless of class, and is used as the argument to other MAVERIK functions. `selectable` notes on a per window basis if the object maintained by SMS are selectable in the usual manner (ie. via keyboard and mouse events). `userdef` can be used to expand this data structure further to include any application specific data.

Applications would rarely need to explicitly create or directly access the fields of this data structure. This should be achieved by the **mav_SMSNew**, **mav_SMSDataGet**, **mav_SMSClassGet**, **mav_SMSSelectabilitySet** functions.

See also

MAV_SMSClass, **MAV_object**, **mav_SMSNew**, **mav_SMSDataGet**, **mav_SMSClassGet**, **mav_SMSSelectabilitySet**

MAV_ctrlF

MAVERIK Level 3 typedefs

Summary

The control function key identifer function type

Syntax

```
typedef void (*MAV_ctrlF)(MAV_window *);
```

Description

MAV_deviceCalcFn

Summary

The device calculate function type

Syntax

```
typedef void (*MAV_deviceCalcFn)(void);
```

Description

MAV_deviceEventFn

MAVERIK Level 3 typedefs

Summary

The device event check function type

Syntax

```
typedef int (*MAV_deviceEventFn)(void);
```

Description

MAV_devicePollFn

Summary

The device poll function type

Syntax

```
typedef void (*MAV_devicePollFn)(void);
```

Description

MAV_moduleIDFn

MAVERIK Level 3 typedefs

Summary

The module identify function type

Syntax

```
typedef char *(*MAV_moduleIDFn)(void);
```

Description

MAV_moduleInitFn

Summary

The module initialization function type

Syntax

```
typedef int (*MAV_moduleInitFn)(void);
```

Description

MAV_texturedObjData

MAVERIK Level 3 typedefs

Summary

Textured object data.

Syntax

```
typedef struct {
  MAV_window *win;
  MAV_object *obj;
  MAV_callbackDrawFn fn;
  MAV_drawInfo di;
  MAV_drawInfo *dip;
  MAV_matrix mat;
} MAV_texturedObjData;
```

Description

On executing the draw callback for a textured object, the **mav_texturedObjectsManage** (page 340) function is called. This routine creates a `MAV_texturedObjData` data structure for the object which contains all the information necessary for it to be rendered at a later date: `win` is the window, `obj` is the object, `fn` the draw callback function, `di` the drawing information to use, `dip` a pointer to `di` or NULL if `di` is not defined, `mat` the state of the graphics matrix.

mav_texturedObjectsManage maintains a list of these data structures (which is reset at the start of each frame). **mav_texturedObjectsRender** (page 340) is called at the end of the frame to traverse this list rendering the objects. Potentially this minimizes the context changes made by the graphics pipe thus increasing performance.

This use of this feature is controlled by the option variable `mav_opt_delayTexture` and is initially disabled.

MAV_transObjData

Summary

Transparent object data.

Syntax

```
typedef struct {
  MAV_window *win;
  MAV_object *obj;
  MAV_callbackDrawFn fn;
  MAV_drawInfo di;
  MAV_drawInfo *dip;
  MAV_matrix mat;
  MAV_BB bb;
  float dist2;
} MAV_transObjData;
```

Description

On executing the draw callback for a transparent object, the **mav_transparentObjectsManage** (page 341) function is called. This routine creates a `MAV_transObjData` data structure for the object which contains all the information necessary for it to be rendered at a later date: `win` is the window, `obj` is the object, `fn` the draw callback function, `di` the drawing information to use, `dip` a pointer to `di` or NULL if `di` is not defined, `mat` the state of the graphics matrix and `dist2` and `bb` the square of the distance from the object to the eye point and its bounding box (needed to depth sort the objects).

mav_transparentObjectsManage maintains a list of these data structures (which is reset at the start of each frame). **mav_transparentObjectsRender** (page 341) is called at the end of the frame to traverse this list depth sorting the objects and rendering them. This ensures that transparent objects are dealt with correctly.

This use of this feature is controlled by the option variable `mav_opt_trans` and is initially disabled. In order for object to take advantage of this feature they must have callback function registered to get their bounding box and surface parameters.

MAV_transTextData

MAVERIK Level 3 typedefs

Summary

Transparent text data.

Syntax

```
typedef struct {
  MAV_window *win;
  char *s;
  int col;
  int font;
  float x;
  float y;
  int justify;
} MAV_transTextData;
```

Description

When drawing a string in a transparent colour, the **mav_transparentTextManage** (page 342) function is called. This routine creates a `MAV_transTextData` data structure for the string which contains all the information necessary for it to be rendered at a later date: `win` is the window, `s` the string, `col` the colour, `font` the font, `x` its horizontal position, `y` its vertical position and `justify` the justification to use.

mav_transparentTextManage maintains a list of these data structures (which is reset at the start of each frame). **mav_transparentTextRender** (page 342) is called at the end of the frame to traverse this list and render the text. This ensures that the transparencies are dealt with correctly.

This use of this feature is controlled by the option variable `mav_opt_trans` and is initially disabled.

Part II

MAVERIK function specifications

Chapter 5

Level 1 functions

---

mav_BBCull

MAVERIK Level 1 functions

mav_BBCull, mav_BBDisplay, mav_BBDisplayWithColour, mav_BBDisplayWithSurfaceParams

---

Summary

Bounding box utility functions.

Syntax

int mav_BBCull(MAV_BB bb);

void mav_BBDisplay(MAV_window *w, MAV_BB bb);

void mav_BBDisplayWithColour(MAV_window *w, MAV_BB bb, int col);

void mav_BBDisplayWithSurfaceParams(MAV_window *w, MAV_BB bb, MAV_surfaceParams *sp);

Description

- **mav_BBCull**
  returns `MAV_TRUE` if bounding box `bb` is inside the current viewing frustum.
- **mav_BBDisplay**
  display the outline of bounding box `bb` in window `w`.
- **mav_BBDisplayWithColour**
  display the outline of bounding box `bb` in window `w` using colour `col`.

- **mav_BBDisplayWithSurfaceParams**
  display the bounding box `bb` in window `w` using surface parameters `sp`.

---

mav_BBPrint

mav_BBPrint, mav_listPrint, mav_matrixPrint, mav_quaternionPrint, mav_surfaceParamsPrint, mav_texCoordPrint, mav_timePrint, mav_timerPrint, mav_vectorPrint, mav_viewParamsPrint, mav_clipPlanePrint, mav_clipPlanesPrint, mav_objectIntersectionPrint, mav_linePrint

---

Summary

Print a data structure.

Syntax

void mav_BBPrint(char *s, MAV_BB bb);

void mav_listPrint(char *s, MAV_list *l);

void mav_matrixPrint(char *s, MAV_matrix m);

void mav_quaternionPrint(char *s, MAV_quaternion q);

void mav_surfaceParamsPrint(char *s, MAV_surfaceParams sp);

void mav_texCoordPrint(char *s, MAV_texCoord t);

void mav_timePrint(char *s, MAV_time t);

void mav_timerPrint(char *s, MAV_timer t);

void mav_vectorPrint(char *s, MAV_vector v);

void mav_viewParamsPrint(char *s, MAV_viewParams vp);

void mav_clipPlanePrint(char *s, MAV_clipPlane cp);

void mav_clipPlanesPrint(char *s, MAV_clipPlanes cp);

void mav_objectIntersectionPrint(char *s, MAV_objectIntersection oi)

void mav_linePrint(char *s, MAV_line ln);

Description

These functions print on `stdout` the contents of the appropriate MAVERIK data structure, preceded by string `s`.

---

mav_SMSDelete

MAVERIK Level 1 functions

Summary

Delete an SMS.

Syntax

void mav_SMSDelete(MAV_SMS *s, int o);

MAV_SMS *s

SMS to be deleted.

int o

Flag specifying whether or not to also delete the contents of the SMS.

Description

This function deletes the specified SMS `s`. If `o` is `MAV_TRUE`, the contents of the SMS are also deleted, otherwise they are not.

---

mav_SMSDisplay

mav_SMSDisplay, mav_SMSDisplayUnCulled

Summary

SMS display functions.

Syntax

int mav_SMSDisplay(MAV_window *w, MAV_SMS *s);

int mav_SMSDisplayUnCulled(MAV_window *w, MAV_SMS *s);

Description

These functions display the contents of SMS `s` in window `w`.

- **mav_SMSDisplay**
  renders the objects, performing the appropriate culling for this type of SMS.
- **mav_SMSDisplayUnCulled**
  renders the objects, but does not perform culling.

The return value of this function is `MAV_TRUE` or `MAV_FALSE` and respectively indicates the success or failure of this operation.

mav_SMSObjListNew

MAVERIK Level 1 functions

mav_SMSObjListNew, mav_SMSHBBNew

Summary

Create an SMS

Syntax

MAV_SMS *mav_SMSObjListNew(void);

MAV_SMS *mav_SMSHBBNew(void);

Description

These function create and return an initialised SMS of the specified type.

- **mav_SMSObjListNew**
  This function creates an "object list" type of SMS. This is the simplest type of SMS and stores objects as a linked list in the order in which they were inserted. View frustum culling is performed when this SMS is displayed by calculating the axis aligned bounding box of each object to determine if it is visible.
- **mav_SMSHBBNew**
  This function create a "hierarchical bounding box" type of SMS. This type of SMS is designed to store the static objects in the scene, i.e. those who's size or position does not change. This restriction means that upon insertion the object's axis aligned bounding box can be calculated and stored for efficiency rather than being re-calculated each time it is required which is the case for the "object list" SMS. Furthermore, as objects are inserted a hierarchy of BB's is built up with the object's BB at the leaf nodes. Testing a BB determines the action required for all the objects beneath it in the hierarchy. View frustum culling is performed when this SMS is displayed by testing the BB's in the hierarchy which indicates either:
    - that all objects beneath it can be removed from further consideration if the BB lies completely outside the view frustum;
    - that all objects beneath it need to be displayed if the BB lies completely inside the view frustum;
    - or, if the BB intersects the frustum, that the BB for the next level down in the hierarchy needs to be checked.

mav_SMSObjectAdd

Summary

Adds an object to an SMS

Syntax

int mav_SMSObjectAdd(MAV_SMS *s, MAV_object *o);

MAV_SMS *s

SMS to add object to.

MAV_object *o

Object to add.

Description

This function adds object `o` to SMS `s`. The return value, set to `MAV_TRUE` or `MAV_FALSE`, indicates whether this operation was successful or not.

In actual fact, this function is nothing more than a wrapper to **mav_SMSCallbackObjectAddExec**. Its purpose is to hide a novice user from a potentially confusing a naming scheme.

See also

mav_SMSObjectRmv, **mav_SMSCallbackObjectAddExec**

mav_SMSObjectRmv

MAVERIK Level 1 functions

Summary

Removes an object from an SMS

Syntax

int mav_SMSObjectRmv(MAV_SMS *s, MAV_object *o);

MAV_SMS *s

SMS to remove object from.

MAV_object *o

Object to remove.

Description

This function removes object `o` from SMS `s`. The return value, set to `MAV_TRUE` or `MAV_FALSE` indicates whether this operation was successful or not.

In actual fact, this function is nothing more than a wrapper to **mav_SMSCallbackObjectRmvExec**. Its purpose is to hide a novice user from a potentially confusing a naming scheme.

See also

mav_SMSObjectAdd, **mav_SMSCallbackObjectRmvExec**

mav_SMSSelectabilitySet

Summary

Change selectability of an SMS.

Syntax

void mav_SMSSelectabilitySet(MAV_SMS *s, MAV_window *w, int v);

MAV_SMS *s

SMS to change.

MAV_window *w

Window.

int v

Whether to make the SMS selectable or not.

Description

This function sets SMS `s` to be selectable (`v = MAV_TRUE`) or non-selectable (`v = MAV_FALSE`) in window `w`. By default, all SMSs are selectable in all windows. The selectability of an SMS also determines whether it is considered by the intersection functions **mav_SMSIntersectLineAll** and **mav_SMSIntersectBBAll**.

---

mav_callbackKeyboardSet

MAVERIK Level 1 functions

mav_callbackKeyboardSet, mav_callbackKeyboardExec

Summary

Keyboard callback management.

Syntax

void mav_callbackKeyboardSet(MAV_window *w, MAV_class *c, MAV_callbackKeyboardFn fn);

int mav_callbackKeyboardExec(MAV_window *w, MAV_object *o,MAV_keyboardEvent *ke);

Description

mav_callbackKeyboardSet sets the callback function for keyboard events for object class `c` in window `w`, to be `fn`.

When a keyboard event is detected by **mav_eventsCheck**, the system checks if the mouse is currently pointing at an object. If it is, and a keyboard callback has been registered for that class of object, the system calls mav_callbackKeyboardExec to execute `fn`, passing it the keyboard event data `me`.

Note: to cause `fn` to be executed regardless of the mouse position (to obtain a non-object specific action) specify object class `o` to be `MAV_CLASS_WORLD`.

See also

mav_eventsCheck (page 136).

---

mav_callbackMouseSet

mav_callbackMouseSet, mav_callbackMouseExec

Summary

Mouse callback management.

Syntax

void mav_callbackMouseSet(int but, MAV_window *w, MAV_class *c,
MAV_callbackMouseFn fn);

int mav_callbackMouseExec(int but, MAV_window *w, MAV_object *o, MAV_mouseEvent *me);

Description

mav_callbackMouseSet sets the callback function for mouse events generated by button `but`, for object class `c` in window `w`, to be `fn`.

When a mouse event is detected by **mav_eventsCheck**, and a mouse callback has been registered for that class of object, the system calls **mav_callbackMouseExec** to execute `fn`, passing it the mouse event data `me`.

See also

mav_eventsCheck (page 136).

---

mav_compositeEmpty

MAVERIK Level 1 functions

Summary

Empty a composite object.

Syntax

void mav_compositeEmpty(MAV_composite *c);

Description

This function empties the composite object `c` that has been read with **mav_compositeReadAC3D**. It deletes the objects it contains and free's up the memory associated with these.

---

mav_compositeRead

mav_compositeRead, mav_compositeReadAC3D, mav_compositeReadAC3DBuf, mav_compositeReadVRML97, mav_compositeReadLWO, mav_compositeReadJIF

Summary

Define a composite object from file.

Syntax

int mav_compositeRead(char *filename, MAV_composite *c, MAV_matrix m);

int mav_compositeReadAC3D(char *filename, MAV_composite *c, MAV_matrix m);

int mav_compositeReadAC3DBuf(char *buf, MAV_composite *c, MAV_matrix m);

int mav_compositeReadVRML97(char *filename, MAV_composite *c, MAV_matrix m);

int mav_compositeReadLWO(char *filename, MAV_composite *c, MAV_matrix m);

int mav_compositeReadJIF(char *filename, MAV_composite *c, MAV_matrix m);

Description

These functions parse geometric models, defined in the various formats, from file, `filename`, or buffer, `buf`, storing the model in composite object `c`.

`mav_compositeRead` examines the extension of `filename` and calls the appropriate parser.

If `mav_opt_compositeSetMatrix` is set to `MAV_TRUE` (its default) then the composite object's matrix is set to be the identity matrix by these functions.

The matrices of various objects which make up of the composite object are multiplied by the matrix `m`. This allows for the object's "default" (that is, when the composite's matrix is an identity matrix) size, orientation and position to be controlled.

The functions returns `MAV_TRUE` on successfully completion, and `MAV_FALSE` otherwise, with the reason for failure printed to `stdout`.

Notes: VRML97 support is only available if MAVERIK has been compiled with this option enabled (see INSTALL file). Only the geometry of VRML97 files is read, no attempt is made to parse scripts, URL's, viewpoints etc... Furthermore, not all of the numerous ways in which the geometry can be defined are supported, e.g. concave polygons, colour-per-vertex. JIF is an now-obsolete format used internally within the Advanced Interfaces Group.

mav_eventsCheck

MAVERIK Level 1 functions

Summary

Check for input device events.

Syntax

int mav_eventsCheck(void);

Description

This function calls the check event function for each registered input device, in the order in which they were registered. If any of the devices' check event functions detects an event has occurred on that device, and an appropriate callback has been registered for that event type, the callback is executed and **mav_eventsCheck** returns immediately.

The return value of the callback forms the return value of this function. If no events are detected the return value is zero. If, and how, this return value is interpreted is an application-specific issue. For example, it could be used to control the drawing of frames. Consider the following three rendering loops:

```
while ( 1 ) {
   if ( mav_eventsCheck() ) {
     /* draw frame */
   }
}

while ( 1 ) {
   mav_eventsCheck();
   /* draw frame */
}

while ( 1 ) {
   while ( mav_eventsCheck() ) {}
   /* draw frame */
}
```

The first loop would only draw a new frame in response to an event occurring for which the callback function returns a true value. The second loop ignores the return value and draws frames in a continuous loop. The third example processes all outstanding events before drawing a frame.

See also

mav_deviceNew

mav_eyeLeft

MAVERIK Level 1 functions

mav_eyeLeft, mav_eyeRight, mav_eyeMono

Summary

Window based view modification functions.

Syntax

void mav_eyeLeft(MAV_window *w);

void mav_eyeRight(MAV_window *w);

void mav_eyeMono(MAV_window *w);

MAV_window *w

Window.

Description

These functions are used as the argument to **mav_windowViewModifierSet**, to set a stereoscopic or monoscopic view in a window.

See also

mav_windowViewModifierSet (page 230)

mav_frameBegin

Summary

Starts a frame.

Syntax

void mav_frameBegin(void);

Description

Calling this function signifies the start of a frame and causes the following actions to occur in order:

- 1. The polling function for each device, defined by **mav_deviceNew**, is called to calculate the device's position in its coordinate frame.
- 2. Any user-defined functions, set with **mav_frameFn0Add**, are executed.
- 3. Any user-defined functions, set with **mav_frameFn1Add**, are executed.
- 4. For each window, the framebuffer is cleared to its background colour (set with **mav_windowBackgroundColourSet**) and the view is set to the values defined by the view parameters associated with the window (set with **mav_windowViewParamsSet**).
- 5. The function to calculate the position of each device in world coordinates is executed.
- 6. Any user-defined functions, set with **mav_frameFn2Add**, are executed.

The purpose of having three sets of user-defined functions is as follows. Those set with **mav_frameFn0Add** or **mav_frameFn1Add** can not perform any rendering, since the framebuffer will not have yet been cleared. However, since the view has not yet been fixed, they can change the values of the view parameters. **mav_frameFn0Add** functions are reserved for use by the navigation routines. Applications should use **mav_frameFn1Add** function if they want to manipulate the view paramaters, e.g. tying the view in one window to related to another.

The opposite is true for user-defined functions set with **mav_frameFn2Add**. They may perform rendering, but since the view has been defined, changing the view parameters will have no effect until the next frame. Furthermore, **mav_frameFn0Add** and **mav_frameFn1Add** functions can only use the local coordinate frame position of any device, whereas **mav_frameFn2Add** functions can use both local and world coordinate frame positions.

See also

mav_frameEnd, **mav_deviceNew**, **mav_frameFn0Add**, **mav_frameFn1Add**, **mav_frameFn2Add**, **mav_windowBackgroundColourSet**, **mav_windowViewParamsSet**.

mav_frameEnd

Summary

indicates the end of a frame.

Syntax

void mav_frameEnd(void);

Description

Calling this function signifies the end of a frame and causes any user-defined functions, set with **mav_frameFn3Add**, to be executed. It then swaps the framebuffers, for all windows in double-buffered configurations, and executes any functions set with **mav_frameFn4Add**.

This function also calculates the wallclock time elapsed between calling **mav_frameBegin** and when the buffers have been swapping by this function. The reciprocal of this value, the frame-rate, is stored in the global variable `mav_fps`.

For high frame rates (short elapsed time) this value will inevitably fluctuate from frame to frame due to variations in system load and the resolution and inaccuracies of the internal clock. The global variable `mav_fps_avg` gives the frame rate averaged over the last second and does not suffer from these problems.

Any rendering that occurs after this function is called, and before the next call to **mav_frameBegin**, will be lost.

See also

mav_frameBegin (page 139), **mav_frameFn3Add** (page 142), **mav_frameFn4Add** (page 142).

---

mav_frameFn0Add

MAVERIK Level 1 functions

mav_frameFn0Add, mav_frameFn1Add, mav_frameFn2Add, mav_frameFn3Add, mav_frameFn4Add, mav_frameFn0Rmv, mav_frameFn1Rmv, mav_frameFn2Rmv, mav_frameFn3Rmv, mav_frameFn4Rmv

---

Summary

Add/remove a function to be called during the render of a frame.

Syntax

void mav_frameFn0Add(MAV_frameFn fn, void *d);

void mav_frameFn1Add(MAV_frameFn fn, void *d);

void mav_frameFn2Add(MAV_frameFn fn, void *d);

void mav_frameFn3Add(MAV_frameFn fn, void *d);

void mav_frameFn4Add(MAV_frameFn fn, void *d);

void mav_frameFn0Rmv(MAV_frameFn fn, void *d);

void mav_frameFn1Rmv(MAV_frameFn fn, void *d);

void mav_frameFn2Rmv(MAV_frameFn fn, void *d);

void mav_frameFn3Rmv(MAV_frameFn fn, void *d);

void mav_frameFn4Rmv(MAV_frameFn fn, void *d);

MAV_frameFn fn

Function to be registered/removed.

Description

mav_frameFnNAdd registers the application-defined function `fn` to be executed at a particular stage of the rendering loop. The various stages of the rendering loop are described in **mav_frameBegin** and **mav_frameEnd**. The `d` parameter is untouched by MAVERIK - it forms the parameter to the application-defined frame function thus allowing argumnets to be passed into the function.

mav_frameFnNRmv remove a previously registered function `fn`.

See also

mav_frameBegin (page 139).

mav_initialise

MAVERIK Level 1 functions

mav_initialise, mav_initialiseNoArgs

Summary

Initialise Maverik.

Syntax

void mav_initialise(int *argc, char *argv[]);

void mav_initialiseNoArgs(void);

Description

These functions are used to initialise MAVERIK and one of them must be the first MAVERIK function called by an application.

Calling either function initialises the kernel and then does one of two things. Either it reads from file `.MavModules` the function names to call to initialise any supporting modules, or, if this file is not present, it initialises the standard set of supporting modules that ship with the MAVERIK distribution.

The `.MavModules` files is searched for in the local directory and the `${MAV_HOME}` directory in that order.

There are a number of ways of defining the various options that control the initialisation process, see the MPG for a full description of these.

These function can also be spelt with a z for our American friends.

mav_keyboardGet

Summary

Sample the keyboard.

Syntax

int mav_keyboardGet(int key);

int key

Keyboard key to sample.

Description

This function samples the state of keyboard key with ASCII code `k`, returning `MAV_PRESSED` if it is currently pressed, otherwise `MAV_RELEASED`.

mav_listItemsAdd

MAVERIK Level 1 functions

mav_listItemsAdd, mav_listItemsRmv, mav_listItemsNext, mav_listItemsContains

Summary

List management functions.

Syntax

void mav_listItemsAdd(MAV_list *l, void *d1, void *d2);

void mav_listItemsRmv(MAV_list *l, void *d1, void *d2);

int mav_listItemsNext(MAV_list *l, void **d1, void **d2);

int mav_listItemsContains(MAV_list *l, void *d1, void *d2);

Description

`MAV_list`'s are actually capable of storing two items of data per entry. The above functions are the two item versions of the list manipulation functions.

`mav_listItemAdd`(l, d) is equivalent to `mav_listItemsAdd`(l, d, NULL).

See also

mav_listNew, **mav_listItemAdd**, **mav_listItemRmv**, **mav_listItemNext**, **mav_listItemContains**

---

mav_listNew

mav_listNew, mav_listItemAdd, mav_listItemRmv, mav_listItemNext, mav_listItemContains, mav_listPointerReset, mav_listPointerPush, mav_listPointerPop, mav_listEmpty, mav_listDelete, mav_listSize

---

Summary

List management functions.

Syntax

MAV_list *mav_listNew(void);

void mav_listItemAdd(MAV_list *l, void *d);

void mav_listItemRmv(MAV_list *l, void *d);

int mav_listItemNext(MAV_list *l, void **d);

int mav_listItemContains(MAV_list *l, void *d);

void mav_listPointerReset(MAV_list *l);

void mav_listPointerPush(MAV_list *l);

void mav_listPointerPop(MAV_list *l);

void mav_listEmpty(MAV_list *l);

void mav_listDelete(MAV_list *l);

int mav_listSize(MAV_list *l);

Description

- **mav_listNew**
  creates a new list, and returns a handle to the list. Lists preserve elements in the order in which they were inserted into the list. `MAV_list`'s are implemented as linked-lists each with its own private "list pointer", and a stack on which to save it, which can be used to conveniently step through the list (see **mav_listItemNext** and related functions).

- **mav_listItemAdd**
  appends item `d` to list `l`.

- **mav_listItemRmv**
  searches list `l` for item `d` and removes it from the list.

- **mav_listItemNext**
  returns, in `d`, the list item currently pointed to by the list pointer of list `l`. The return value of the function is `MAV_TRUE` if the data was successfully returned, otherwise `MAV_FALSE`. The pointer is then moved onto the next item.

- **mav_listItemContains**
  searches list `l` for item `d`. The function returns `MAV_TRUE` if the the item is located, otherwise `MAV_FALSE`.

- **mav_listPointerReset**
  sets the list pointer of list `l` to point to the beginning of the list.

- **mav_listPointerPush**
  This function pushes the list pointer for list `l` onto its stack. The value of the list pointer is unchanged.

- **mav_listPointerPop**
  pops the list pointer for list `l` from its stack.

- **mav_listEmpty**
  deletes all the linked-list nodes from list `l`. It does not, however, delete the data referenced by the nodes in the list. After calling this function, `l` refers to an empty list, which can be used again.

- **mav_listDelete**
  deletes all the nodes in list `l`. It does not, however, delete any data referenced by the nodes in the list. After calling this function, `l` is undefined.

- **mav_listSize**
  returns the number of items in list `l`.

---

mav_malloc

mav_malloc, mav_calloc, mav_free

Summary

Maverik dynamic memory management.

Syntax

void *mav_malloc(int amount);

void *mav_calloc(int nelem, int elemsize);

void mav_free(void *d);

Description

For convenience, MAVERIK provides wrapper functions to the standard system **malloc**, **calloc** and **free** functions. The wrappers check for out-of-memory errors.

mav_malloc attempts to malloc `amount` bytes of memory, and if successful returns a pointer to the base of the allocated memory. If unsuccessful, it calls **exit**(1).

mav_calloc performs the same function as the system **calloc** but provides the error-checking described above.

mav_free calls **free** with argument `d`.

---

mav_matrixInverse

MAVERIK Level 1 functions

Summary

Invert a matrix.

Syntax

MAV_matrix mav_matrixInverse(MAV_matrix m);

MAV_matrix m

Matrix to invert.

Description

This function computes the inverse of matrix `m`, and returns the result. If the matrix is singular, the function prints an error message.

---

mav_matrixMult

Summary

Multiply two matrices.

Syntax

MAV_matrix mav_matrixMult(MAV_matrix m1, MAV_matrix m2);

MAV_matrix m1

Matrix to multiply.

MAV_matrix m2

Matrix to multiply.

Description

This function computes `m1 *m2`, returning the result.

mav_matrixQuaternionConvert

MAVERIK Level 1 functions

Summary

Create a matrix from a quaternion.

Syntax

MAV_matrix mav_matrixQuaternionConvert(MAV_quaternion q);

MAV_quaternion q

Quaternion.

Description

This function returns a matrix corresponding to the quaternion `q`.

mav_matrixRPYGet

Summary

Query rotation terms of matrix.

Syntax

void mav_matrixRPYGet(MAV_matrix m, float *r, float *p, float *y);

MAV_matrix m

Matrix to query.

float *r

Roll in degrees.

float *p

Pitch in degrees.

float *y

Yaw in degrees.

Description

This function returns in `r`, `p` and `y`, the roll, pitch and yaw specifed by the rotation elements of matrix `m`. `m` must be of unit scale.

The conversion from an orientation matrix to a set of roll, pitch and yaw values is inherently ill-defined. That is to say there are multiple sets of RPY values which describe a given orientation - there is no one-to-one mapping. For example, a RPY of (0, 0, 145) is mathematically identical to one of (180, 180, 35) in that they both give the same orientation.

mav_matrixRPYGet returns just one of the many possible RPY values which can describe a given orientation. This may or may not be the most “intuitive” set.

If you use all three RPY values together there should not be a problem. What you cant do is modify one of the values in isolation and expect to get sensible behaviour.

mav_matrixRPYSet

MAVERIK Level 1 functions

Summary

Set the rotation terms of a matrix.

Syntax

MAV_matrix mav_matrixRPYSet(MAV_matrix m, float roll, float pitch, float yaw);

MAV_matrix m

Matrix to set.

float roll

Rotation in degrees about Z-axis.

float pitch

Rotation in degrees about X-axis.

float yaw

Rotation in degrees about Y-axis.

Description

This function sets the rotation elements of matrix `m`, as specified by `roll`, `pitch` and `yaw`. All other elements in matrix `m` are left unchanged, and the modified matrix is returned.

mav_matrixScaleSet

Summary

Scale the elements of a matrix.

Syntax

MAV_matrix mav_matrixScaleSet(MAV_matrix m, float sc);

MAV_matrix m

Matrix to scale.

float sc

Scale factor.

Description

This function multiplies the rotation and scaling terms of `m` by `sc`. The translation terms are not scaled.

mav_matrixSet

MAVERIK Level 1 functions

Summary

Compute a matrix for orientation and position.

Syntax

MAV_matrix mav_matrixSet(float roll, float pitch, float yaw, float x, float y, float z);

float roll

Rotation in degrees about Z-axis.

float pitch

Rotation in degrees about X-axis.

float yaw

Rotation in degrees about Y-axis.

float x

Translation in x.

float y

Translation in y.

float z

Translation in z.

Description

This function defines a matrix to perform a rotation followed by a translation. The rotation angles in degrees are defined by `roll` (around the Z-axis), pitch (around the X-axis), and `yaw` (around the Y-axis). Rotations are applied in the order roll, yaw, then pitch. The translation, relative to the origin, is defined by `x`, `y` and `z`.

---

mav_matrixXAxisGet

mav_matrixXAxisGet, mav_matrixYAxisGet, mav_matrixZAxisGet

Summary

Return the various axes of a matrix.

Syntax

MAV_vector mav_matrixXAxisGet(MAV_matrix m);

MAV_vector mav_matrixYAxisGet(MAV_matrix m);

MAV_vector mav_matrixZAxisGet(MAV_matrix m);

MAV_matrix m

Matrix in question.

Description

These functions return the axes of matrix `m`, i.e. the result of multiplying the principle axis by `m`. This will be the vector the axis will map onto if `m` is used to transform coordinate systems.

---

mav_matrixXAxisSet

MAVERIK Level 1 functions

mav_matrixXAxisSet, mav_matrixYAxisSet, mav_matrixZAxisSet

Summary

Set the various axes of a matrix.

Syntax

MAV_matrix mav_matrixXAxisSet(MAV_matrix m, MAV_vector v);

MAV_matrix mav_matrixYAxisSet(MAV_matrix m, MAV_vector v);

MAV_matrix mav_matrixZAxisSet(MAV_matrix m, MAV_vector v);

MAV_matrix m

Matrix in question.

MAV_vector v

Vector to use.

Description

These functions set the axes of matrix `m` returning the result, i.e. they set the result of multiplying the principle axis by `m`. This will be the vector the axis will map onto if `m` is used to transform coordinate systems. N.B. Mathematics dictates that axes can not be set independently of one another.

---

mav_matrixXYZGet

Summary

Query translation terms of matrix.

Syntax

MAV_vector mav_matrixXYZGet(MAV_matrix m);

MAV_matrix m

Matrix to query.

Description

This function returns the translation terms of matrix `m`.

---

mav_matrixXYZSet

Summary

Set the translation terms of a matrix.

Syntax

MAV_matrix mav_matrixXYZSet(MAV_matrix m, MAV_vector v);

MAV_matrix m

Matrix to set.

MAV_vector v

3D vector to specify translation.

Description

This function sets the translation terms of matrix `m`, as specified by vector `v`. All other terms in matrix `m` are left unchanged, and the modified matrix is returned.

---

mav_mouseDraw

Summary

Draw mouse cursor.

Syntax

void mav_mouseDraw(void *ignored);

Description

This function draws a 2D cross on the screen corresponding the current mouse position.

mav_mouseGet

Summary

Sample the current mouse position and button status.

Syntax

void mav_mouseGet(MAV_window *w, int *x, int *y, int *rx, int *ry, int *buts);

MAV_window *w

Window.

int *x, int *y

returns (x, y) position of mouse relative to window origin (pixels).

int *rx, int *ry

returns (x, y) position of mouse relative to root window origin (pixels).

int *buts

returns button status.

Description

This function returns the current mouse position (`x`,`y`) in window `w`, and with respect to the root (background) window (`rx`,`ry`). If non-NULL, the array `buts` is filled with the status (`MAV_PRESSED` or `MAV_RELEASED`) of each button.

mav_mouseSurfaceParamsSet

Summary

Sets the colour with which to draw the mouse cursor.

Syntax

void mav_mouseSurfaceParamsSet(MAV_surfaceParams *sp);

MAV_surfaceParams *sp

The surface parameters to use.

Description

This function sets the colour used by **mav_mouseDraw**. Since the mouse is rendered as lines, an emmisive colour is the only sensible set of surface parameters which can be used.

---

mav_navigateNull

MAVERIK Level 1 functions

mav_navigateNull, mav_navigateTransX, mav_navigateTransY, mav_navigateTransZ, mav_navigateRotRight, mav_navigateRotUp, mav_navigateForwards, mav_navigateForwardsFixedUp, mav_navigateUp, mav_navigateUpFixedUp, mav_navigateRight, mav_navigateRightFixedUp, mav_navigateRoll, mav_navigatePitch, mav_navigatePitchFixedUp, mav_navigateYaw, mav_navigateYawFixedUp

---

Summary

Navigation functions.

Syntax

void mav_navigateNull(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateTransX(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateTransY(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateTransZ(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateRotRight(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateRotUp(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateForwards(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateForwardsFixedUp(MAV_viewParams *vp, float am, float ls,float as);

void mav_navigateUp(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateUpFixedUp(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateRight(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateRightFixedUp(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateRoll(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigatePitch(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigatePitchFixedUp(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateYaw(MAV_viewParams *vp, float am, float ls, float as);

void mav_navigateYawFixedUp(MAV_viewParams *vp, float am, float ls, float as);

Description

These are the navigation functions:

- **mav_navigateNull**
  does nothing.

- **mav_navigateTransX**
  translates the eyepoint along the world x-axis by an amount `am` *`ls`.

- **mav_navigateTransY**
  translates the eyepoint along the world y-axis by an amount `am` *`ls`.

- **mav_navigateTransZ**
  translates the eyepoint along the world z-axis by an amount `am` *`ls`.

- **mav_navigateRotRight**
  rotates the view direction vectors and the eyepoint about the view right vector by amount `am` *`as`. `ls` is ignored.

- **mav_navigateRotUp**
  rotates the view direction vectors, and the eyepoint about the view up vector by amount `am` *`as`. `ls` is ignored.

- **mav_navigateForwards**
  moves the eypoint forwards along the view direction vector by an amount `am` *`ls`.

- **mav_navigateForwardsFixedUp**
  moves the eypoint along the projection of the view vector onto the place normal to the global up vector, by an amount `am` *`ls`.

- **mav_navigateUp**
  moves the eypoint along the view up vector by an amount `am` *`ls`.

- **mav_navigateUpFixedUp**
  moves the eypoint along the world “up” vector by an amount `am` *`ls`.

- **mav_navigateRight**
  moves the eypoint along the view right vector by an amount `am` *`ls`.

- **mav_navigateRightFixedUp**
  moves the eypoint along the projection of the view right vector onto the place normal to the global up vector by an amount `am` *`ls`.

- **mav_navigateRoll**
  rotates the view vectors about the view direction vector by an amount `am` * `as`.

- **mav_navigatePitch**
  rotates the view vectors about the view right vector by an amount `am` * `as`.

- **mav_navigatePitchFixedUp**
  rotates the view vectors about the projection of the view right vector onto the plane normal to the global up vector, by an amount `am` * `as`.

- **mav_navigateYaw**
  rotates the view vectors about the view up vector by an amount `am` * `as`.

- **mav_navigateYawFixedUp**
  rotates the view vectors about the world up vector by an amount `am` * `as`.

mav_navigationKeyboardDefaultParams

Summary

Provides control of the default implementation of keyboard navigation

Syntax

void mav_navigationKeyboardDefaultParams(MAV_window *w, float am, float ls, float as);

Description

This function defines the linear and angular scaling factors (`ls` and `as`) used by the default keyboard navigation. The value `am` can be thought of as the amount of movement a key gives (this value is multiplied by the appropriate scaling factor to give the true movement). Setting `am` to 50 (its default value) makes a navigation function invoked by the keyboard equivalent to it being invoked by 50 pixels of mouse movement.

mav_navigationKeyboardDefault

MAVERIK Level 1 functions

Summary

The default implementation of keyboard navigation

Syntax

int mav_navigationKeyboardDefault(MAV_object *o, MAV_keyboardEvent *e);

Description

This function is the default implementation of keyboard navigation and is used as the second argument to **mav_navigationKeyboard**, i.e.

```
mav_navigationKeyboard(mav_win_all, mav_navigationKeyboardDefault);
```

It provides the following "Doom" style controls:

- Cursor keys – navigate forwards/backwards and yaw;
- Page up/down – navigate up/down;
- Alt-Cursor left/right – sidestep left/right;
- Alt-Page up/down – pitch view up/down;
- Holding down "shift" doubles the rate of movement.

Linear translations assume that the application is using meters as its units.

Control of the default keyboard navigation, via **mav_navigationKeyboardDefaultParams**, is more limited that the mouse variety since you can't redefine the actions taken by the various keys.

See also

mav_navigationKeyboard, **mav_navigationKeyboardDefaultParams**, **mav_navigationMouseDefault**

---

mav_navigationKeyboard

MAVERIK Level 1 functions

Summary

Keyboard navigation management.

Syntax

void mav_navigationKeyboard(MAV_window *w, MAV_callbackKeyboardFn fn);

Description

This routines sets the function to implement keyboard navigation in window `w` to be `fn`.
MAVERIK provides a default implementation of keyboard navigation with the function **mav_navigationKeyboardDefault**.

See also

mav_navigationKeyboardDefault, **mav_navigationMouse**

mav_navigationMouseDefaultParams

Summary

Provides control of the default implementation of mouse navigation

Syntax

void mav_navigationMouseDefaultParams(MAV_window *w, int but, MAV_navigatorFn x, float xls, float xas, MAV_navigatorFn y, float yls, float yas);

Description

Mouse navigation is controlled on a per-window (`w`) and per-button (`but`) basis and consists of two sets of three values, the first for horizontal mouse movements, the second for vertical. The three arguments in each set are:

- `x` (and `y`): a function (of type **MAV_navigatorFn**) which performs the required navigation. MAVERIK provides many default navigator functions.
- `xls` (and `yls`): the scaling factor to convert from pixels into application units in order to apply linear movements.
- `xas` (and `yas`): the scaling factor to convert from pixels into radians in order to apply angular movements (this is usually independent of the application).

For example, the following defines navigation triggered by the left mouse button. Horizontal movements of the mouse yaws the view; and vertical movement moves the view forward:

```
mav_navigationMouseDefaultParams(mav_win_all, MAV_LEFT_BUTTON,
                                 mav_navigateYaw, 0.02, -0.001,
                                 mav_navigateForwards, 0.02, 0.001);
```

A vertical mouse movement of 100 pixels equates to the eyepoint moving 2 (100x0.02) application units forwards.

Note that the angular scaling factor is negative for **mav_navigateYaw**. This is because a right-handed coordinate system is assumed which implies that a positive yaw will rotate the view to the left. This is the opposite to what is required, and so a negative scaling factor is used to compensate.

See also

MAV_navigatorFn, default navigator functions

mav_navigationMouseDefault

Summary

The default implementation of mouse navigation

Syntax

int mav_navigationMouseDefault(MAV_object *o, MAV_mouseEvent *e)

Description

This function is the default implementation of mouse navigation and is used as the second argument to **mav_navigationMouse**, i.e.

```
mav_navigationMouse(mav_win_all, mav_navigationMouseDefault);
```

It provides the following controls:

- with the left button pressed, mouse movements translate the eyepoint forward/backwards, and yaws (rotates about the Y axis) the view.
- with the right button pressed, mouse movements translate the eyepoint up/down and left/right.

Linear translations assume that the application is using meters as its units.

The style of mouse navigation provided by this function is fully configurable via **mav_navigationMouseDefaultParams**.

See also

mav_navigationMouse, **mav_navigationMouseDefaultParams**, **mav_navigationKeyboardDefault**

mav_navigationMouse

MAVERIK Level 1 functions

Summary

Mouse navigation management.

Syntax

void mav_navigationMouse(MAV_window *w, MAV_callbackMouseFn fn);

Description

This routines sets the function to implement mouse navigation in window `w` to be `fn`.

MAVERIK provides a configurable default implementation of mouse navigation with the function **mav_navigationMouseDefault**.

See also

mav_navigationMouseDefault, **mav_navigationKeyboard**

mav_objectClassGet

Summary

Query the class of an object.

Syntax

MAV_class *mav_objectClassGet(MAV_object *o);

MAV_object *o

Pointer to object to query.

Description

Returns the class of object `o`, i.e. the class used to create the object by **mav_objectNew**.

See also

mav_objectNew

mav_objectDataGet

MAVERIK Level 1 functions

Summary

Query the data associated with an object.

Syntax

void *mav_objectDataGet(MAV_object *o);

MAV_object *o

Object to query.

Description

This function returns a pointer to the data associated with object `o`, i.e. the data used to create the object by **mav_objectNew**.

By necessity the data must be a void pointer so caution must be taken when dereferencing it to a specfic data type.

See also

mav_objectNew, **mav_objectClassGet**, **mav_objectDataWith**,

mav_objectDataWith

Summary

Query the object bound to the specified data.

Syntax

MAV_object *mav_objectDataWith(void *d);

void *d

Data to query.

Description

This function efficiently searches through all known MAVERIK objects to locate the object whose associated data is `d`.

See also

mav_objectNew, **mav_objectDataGet**

mav_objectDelete

MAVERIK Level 1 functions

Summary

Delete an object.

Syntax

void mav_objectDelete(MAV_object *o);

MAV_object *o

Object to delete.

Description

This function deletes the object specified by `o`. The object will automatically be removed from any SMS's it is in, and, if set, have the callback function (set with **mav_callbackDeleteSet**) executed. Typically this callback function will be used to free up any memory used by the object.

See also

mav_callbackDeleteSet

mav_objectNew

Summary

Create a new object.

Syntax

MAV_object *mav_objectNew(MAV_class *c, void *d);

MAV_class *c

Class of object.

void *d

Pointer to data associated with the object.

Description

This function creates a new object. It encapsulates in a single data structure (the resultant **MAV_object**) the methods defined by class `c`, and the data defined by `d`. **MAV_object**'s acts as generic handle to any object, regardless of class, and is used as the argument to other MAVERIK functions.

The **MAV_object** created by this function contains a reference to – not a copy of – the data and class. This means that if the data or class change in the future, the relevant **MAV_object**(s) do not need to be notified.

See also

MAV_object, **mav_objectDelete**, **mav_objectDataGet**, **mav_objectClassGet**, **mav_objectDataWith**

mav_paletteColourIndexEmptyGet

MAVERIK Level 1 functions

mav_paletteColourIndexEmptyGet, mav_paletteMaterialIndexEmptyGet, mav_paletteTextureIndexEmptyGet, mav_paletteLightIndexEmptyGet, mav_paletteFontIndexEmptyGet

Summary

Return an empty index in the palette.

Syntax

int mav_paletteColourIndexEmptyGet(MAV_palette *p);

int mav_paletteMaterialIndexEmptyGet(MAV_palette *p);

int mav_paletteTextureIndexEmptyGet(MAV_palette *p);

int mav_paletteLightIndexEmptyGet(MAV_palette *p);

int mav_paletteFontIndexEmptyGet(MAV_palette *p);

MAV_palette *p

Palette to query

Description

These functions return the first empty index for the relevant component in a given palette. If no empty index can be found a warning message a printed and -1 returned.

mav_paletteColourIndexMatchGet

mav_paletteColourIndexMatchGet, mav_paletteMaterialIndexMatchGet, mav_paletteTextureIndexMatchGet, mav_paletteLightIndexMatchGet, mav_paletteFontIndexMatchGet

Summary

Return a matching index in the palette.

Syntax

int mav_paletteColourIndexMatchGet(MAV_palette *p, float ar, float ag, float ab, float aa);

int mav_paletteMaterialIndexMatchGet(MAV_palette *p, float ar, float ag, float ab, float aa, float dr, float dg, float db, float da, float sr, float sg, float sb, float sa, float er, float eg, float eb, float ea, float shin);

int mav_paletteTextureIndexMatchGet(MAV_palette *p, char *name);

int mav_paletteLightIndexMatchGet(MAV_palette *p, float ar, float ag, float ab, float aa, float dr, float dg, float db, float da, float sr, float sg, float sb, float sa);

int mav_paletteFontIndexMatchGet(MAV_palette *p, char *name);

MAV_palette *p

Palette to query

float ar, ag, ab, aa

Ambient R, G, B, A (0.0-1.0).

float dr, dg, db, da

Diffuse R, G, B, A (0.0-1.0).

float sr, sg, sb, sa

Specular R, G, B, A (0.0-1.0).

float er, eg, eb, ea

Emissive R, G, B, A (0.0-1.0).

float shin

Shininess (specular Phong exponent) (0.0-128.0)

char *name

File or font name

Description

These functions return the index of the first matching relevant component in a given palette. If no matching index can be found -1 returned.

mav_paletteColourSet

Summary

Define an ambient colour entry in a palette.

Syntax

**void mav_paletteColourSet(MAV_palette *p, int index, float r, float g, float b,
float a);**

MAV_palette *p

Palette to set.

int index

Index of colour to define, in range 0..`mav_opt_maxColours` (default 150).

float r, float g, float b

Red, green and blue component of colour (0.0-1.0)

float a

Alpha component of colour (0.0-1.0) (0=transparent, 1=opaque)

Description

This function defines ambient colour entry `index` of palette `p` as specified by `r`, `g`, `b` and `a`.

N.B. For transparent colours to be handled correctly, the options variable `mav_opt_trans` must be set to `MAV_TRUE` before **mav_initialise** is called.

See also

mav_paletteMaterialSet, **mav_paletteTextureSet**

mav_paletteFontSet

MAVERIK Level 1 functions

Summary

Define a font entry in a palette.

Syntax

void mav_paletteFontSet(MAV_palette *p, int index, char *s);

MAV_palette *p

Palette to set.

int index

Index of font to define

char *s

Name of font to define

Description

mav_paletteLightPos

Summary

Set palette light position.

Syntax

void mav_paletteLightPos(MAV_palette *p, int index, MAV_vector pos);

MAV_palette *p

Palette to set.

int index

Index into lights table, in range 0..`mav_opt_maxLights` (default 5).

MAV_vector pos

Position vector.

Description

This function sets the position of the light at entry `index` in the light table of palette `p` to be `pos`. By default lights are positioned relative to the eye point and, once defined, follow the eyepoint to give a car-headlight effect. Use `mav_paletteLightPositioning` to specify that a light should have an absolute fixed position defined in world coordinates.

mav_paletteLightPositioning

MAVERIK Level 1 functions

Summary

Set palette light positioning.

Syntax

void mav_paletteLightPositioning(MAV_palette *p, int index, int pos);

MAV_palette *p

Palette to set.

int index

Index into lights table, in range `0..mav_opt_maxLights` (default 5).

int pos

Positioning strategy.

Description

This function sets the positioning strategy of the light at entry `index` in the light table of palette `p` to be `pos`.

Acceptable values of `pos` are:

- `MAV_LIGHT_RELATIVE` (the default)
  to specify that the light is to be positioned relative to the eye, and subsequently follow it, to give a car-headlight effect.
- `MAV_LIGHT_ABSOLUTE`
  to specify that the light is to be positioned in world coordinates to give the effect of a light at a fixed position in the model.

mav_paletteLightSet

Summary

Define a light.

Syntax

void mav_paletteLightSet(MAV_palette *p, int index, float ar, float ag, float ab, float aa, float dr, float dg, float db, float da, float sr, float sg, float sb, float sa);

MAV_palette *p

Palette to set.

int index

Index into lights table, in range 0..`mav_opt_maxLights` (default 5).

float ar, float ag, float ab, float aa

RGBA components of ambient colour (0.0 - 1.0)

float dr, float dg, float db, float da

RGBA components of diffuse colour (0.0 - 1.0)

float sr, float sg, float sb, float sa

RGBA components of specular colour (0.0 - 1.0)

Description

This function sets the colour specification of the light at entry `index` in the light table of palette `p`.

mav_paletteLightingModelSet

MAVERIK Level 1 functions

Summary

Set palette lighting model.

Syntax

void mav_paletteLightingModelSet(MAV_palette *p, float ar, float ag, float ab, float aa, int local);

MAV_palette *p

Palette to set.

float ar, float ag, float ab, float aa

RGBA components of ambient colour (0.0 - 1.0)

int local

Local or infinite viewpoint.

Description

This function sets the lighting model for palette `p`. The ambient component of the light is given by (`ar`, `ag`, `ab`, `aa`). `local` specifies whether the viewpoint is local (`local` = `MAV_TRUE`) or inifinite (`local` = `MAV_FALSE`).

mav_paletteMaterialSet

Summary

Define a material entry in a palette.

Syntax

**void mav_paletteMaterialSet(MAV_palette *p, int index, float ar, float ag, float ab,
float aa, float dr, float dg, float db, float da, float sr, float sg, float sb,
float sa, float er, float eg, float eb, float ea, float shin);**

MAV_palette *p

Palette to set.

int index

Index of materials to define, in range 0..`mav_opt_maxMaterials` (default 150).

float ar, ag, ab, aa

Ambient R, G, B, A (0.0-1.0).

float dr, dg, db, da

Diffuse R, G, B, A (0.0-1.0).

float sr, sg, sb, sa

Specular R, G, B, A (0.0-1.0).

float er, eg, eb, ea

Emissive R, G, B, A (0.0-1.0).

float shin

Shininess (specular Phong exponent) (0.0-128.0)

Description

This function defines material entry `index` of palette `p` as specified by the various values described above.

See also

mav_paletteColourSet, **mav_paletteTextureSet**

mav_paletteTextureAlphaSet

Summary

Set the alpha component of a palette texture.

Syntax

void mav_paletteTextureAlphaSet(MAV_palette *p, int index, float a);

MAV_palette *p

Palette to set.

int index

Index into textures table, in range 0..`mav_opt_maxTextures` (default 150).

float a

Alpha value (0.0-1.0).

Description

This function sets the alpha component of the texture in entry `index` of the texture table in palette `p`. `alpha` is in the range 0 (fully transparent) to 1 (fully opaque).

mav_paletteTextureColourAlphaSet

MAVERIK Level 1 functions

Summary

Set the alpha component for a colour in a palette texture.

Syntax

void mav_paletteTextureColourAlphaSet(MAV_palette *p, int index, int r, int g, int b, float a);

MAV_palette *p

Palette to set.

int index

Index into textures table, in range 0..`mav_opt_maxTextures` (default 150).

int r, int g, int b

RGB of colour to match (each 0-255).

float a

Alpha value (0.0 - 1.0)

Description

This function is similar to **mav_paletteTextureAlphaSet**. However, it only sets the alpha components of those colours in the texture whose RGB values exactly match `r`, `g` and `b`.

See also

mav_paletteTextureAlphaSet (page 191).

mav_paletteTextureEnvPaletteSet

Summary

Set a palette's texture environment function.

Syntax

void mav_paletteTextureEnvPaletteSet(MAV_palette *p, MAV_texEnvFn fn);

MAV_palette *p

Palette.

MAV_texEnvFn fn

Texture environment function.

Description

This function sets the texture environment function for all the entries in palette `p`. The texture environment function controls the way textures are rendered (blend, decal, modulate or lit). Note that the setting can be overridden on a per-texture basis using **mav_paletteTextureEnvSet**.

See also

mav_paletteTextureEnvSet (page 334).

mav_paletteTextureFree

MAVERIK Level 1 functions

Summary

Free a palette texture.

Syntax

void mav_paletteTextureFree(MAV_palette *p, int index);

MAV_palette *p

Palette.

int index

Index into textures table, in range 0..`mav_opt_maxTextures` (default 150).

Description

This function frees the memory allocated for the texture stored at entry `index` in the texture table of palette `p`.

See also

mav_paletteTextureSetFromMem (page 196).

mav_paletteTextureMipmappingSet

Summary

Set the mipmapping of a palette texture.

Syntax

void mav_paletteTextureMipmappingSet(MAV_palette *p, int index, int v);

MAV_palette *p

Palette to set.

int index

Index into textures table, in range 0..`mav_opt_maxTextures` (default 150).

int v

Mipmapping enabled or not.

Description

This function controls the mipping of the texture in entry `index` of the texture table in palette `p`. Setting `v` to `MAV_TRUE` enables mipmapping, setting its value to `MAV_FALSE` disabled mipmapping. This function overrides the global mipmapping default, defined by `mav_opt_mipmapping`, for a specfic texture. If a texture is to be mipmapped then this must be specified before the texture is defined.

mav_paletteTextureSetFromMem

MAVERIK Level 1 functions

Summary

Set a palette texture from a texture in memory.

Syntax

int mav_paletteTextureSetFromMem(MAV_palette *p, int index, int width, int height, unsigned long *mem);

MAV_palette *p

Palette to set.

int index

Index into textures table, in range 0..`mav_opt_maxTextures` (default 150).

int width

Width of texture, in pixels.

int height

Height of texture, in pixels.

unsigned long *mem

Pointer to start of memory holding texture.

Description

This function sets the texture for entry `index` in the textures table of palette `p`. A texture of size (`width` x `height`) is obtained from memory starting at address `mem`. Note that the actual texture is not copied into the texture table, only referenced.

See also

mav_paletteTextureFree (page 194).

mav_paletteTextureSet

Summary

Define a texture entry in a palette.

Syntax

int mav_paletteTextureSet(MAV_palette *p, int index, char *filename);

MAV_palette *p

Palette to set.

int index

Index of texture to define, in range 0..`mav_opt_maxTextures` (default 150);

char *filename

The name of the file containing the texture.

Description

This function defines texture entry `index` of palette `p` from file `filename`. MAVERIK itself only supports the PPM (raw or ASCII encodings) file format for textures. However, MAVERIK can use the ImageMagick `convert` program, if installed, to convert almost any other image file format into PPM and then parse that. This conversion process is hidden from the user so that MAVERIK appears to support virtually all image file formats. Furthermore, MAVERIK uses ImageMagick's `convert` program to resize the image, if needed, so that it is an integer power of 2 in both width and height – a requirement placed on texture images by OpenGL.

The function returns `MAV_TRUE` if the texture was successfully read and `MAV_FALSE` if the operation failed.

See also

mav_paletteColourSet, **mav_paletteMaterialSet**

---

mav_paletteWarn

MAVERIK Level 1 functions

mav_paletteWarnSet, mav_paletteColourWarnSet, mav_paletteMaterialWarnSet, mav_paletteTextureWarnSet, mav_paletteLightWarnSet, mav_paletteLightingModelWarnSet, mav_paletteFontWarnSet, mav_paletteColourIndexWarnSet, mav_paletteMaterialIndexWarnSet, mav_paletteTextureIndexWarnSet, mav_paletteLightIndexWarnSet, mav_paletteFontIndexWarnSet

---

Summary

Define whether to warn on redefinition or not

Syntax

void mav_paletteWarnSet(MAV_palette *p, int v);

void mav_paletteColourWarnSet(MAV_palette *p, int v);

void mav_paletteMaterialWarnSet(MAV_palette *p, int v);

void mav_paletteTextureWarnSet(MAV_palette *p, int v);

void mav_paletteLightWarnSet(MAV_palette *p, int v);

void mav_paletteLightingModelWarnSet(MAV_palette *p, int v);

void mav_paletteFontWarnSet(MAV_palette *p, int v);

void mav_paletteColourIndexWarnSet(MAV_palette *p, int idx, int v);

void mav_paletteMaterialIndexWarnSet(MAV_palette *p, int idx, int v);

void mav_paletteTextureIndexWarnSet(MAV_palette *p, int idx, int v);

void mav_paletteLightIndexWarnSet(MAV_palette *p, int idx, int v);

void mav_paletteFontIndexWarnSet(MAV_palette *p, int idx, int v);

MAV_palette *p

Palette to set

int idx

Colour, material, texture, light or font index

int v

Whether to warn on redefinition or not

Description

These functions define whether or not a warning is generated if the contents of a palette are redefined. This warning can be on a per-palette basis, per-type (colour, material, texture, light or font) or per-type-index basis.

mav_quaternionInterpolate

MAVERIK Level 1 functions

Summary

Interpolates a quaternion.

Syntax

MAV_quaternion mav_quaternionInterpolate(MAV_quaternion st, MAV_quaternion fi, float val);

MAV_quaternion st

Start quaternion.

MAV_quaternion fi

Finish quaternion.

float val

Interpolation constant in the range (0.0–1.0).

Description

This function performs spherical linear interpolation between quaternions `st` and `fi`, returning the result.

---

mav_quaternionMatrixConvert

Summary

Create a quaternion from a matrix.

Syntax

MAV_quaternion mav_quaternionMatrixConvert(MAV_matrix m);

MAV_matrix m

Matrix.

Description

This function converts a 4x4 transformation matrix into a quaternion.

---

mav_quaternionSet

MAVERIK Level 1 functions

Summary

Define a quaternion.

Syntax

MAV_quaternion mav_quaternionSet(MAV_vector ax, float ang);

MAV_vector ax

3D axis of rotation.

float ang

Rotation angle in degrees.

Description

This function returns the quaternion corresponding to the specified rotation `ang` about the specified axis `ax`, which passes through the origin.

---

mav_random

mav_random, mav_randomSeed

Summary

Random number functions.

Syntax

float mav_random(void);

void mav_randomSeed(long seed);

Description

- **mav_random**
  returns a random number in the range 0.0-1.0.
- **mav_randomSeed**
  sets the seed for the random number generator to be `seed`, unless `seed` is negative, in which case the seed is set to the current system time.

---

mav_sleep

MAVERIK Level 1 functions

Summary

Sleep.

Syntax

void mav_sleep(float len);

float len

Time to sleep (seconds).

Description

This function waits for `len` seconds, which can be specified with microsecond resultion.

---

mav_stringDisplay

mav_stringDisplay, mav_stringDisplayLeft, mav_stringDisplayCentre, mav_stringDisplayRight

Summary

Display annotation text in a window.

Syntax

void mav_stringDisplay(MAV_window *w, char *s, int col, int font, float x, float y);

void mav_stringDisplayLeft(MAV_window *w, char *s, int col, int font, float x, float y);

void mav_stringDisplayCentre(MAV_window *w, char *s, int col, int font, float x, float y);

void mav_stringDisplayRight(MAV_window *w, char *s, int col, int font, float x, float y);

MAV_window *w

Window.

char *s

String to display.

int col

Colour of text.

int font

Font of text.

float x, float y

(x,y) position of text in NDC (-1.0 to 1.0).

Description

These functions display string `s` in window `w` at position `x`,`y` with relevant justification in colour `col` and font `font`. `mav_stringDisplay` produces the same results as `mav_stringDisplayLeft`.

These functions are intended for displaying “annotation” text, which remains “glued” to the NDC plane and always legible (provided the left most edge of the text is within the window).

mav_stringLength

MAVERIK Level 1 functions

Summary

Calculate length of annotation text.

Syntax

int mav_stringLength(MAV_window *w, char *s, int font);

MAV_window *w

Window.

char *s

String to calculate length of.

int font

Font of text.

Description

This function calculates and returns the length (in pixels) of string `s` when displayed in font `font` in window `w`.

mav_surfaceParamsNew

Summary

Create a new set of surface parameters.

Syntax

MAV_surfaceParams *mav_surfaceParamsNew(int mode, int colour, int material, int texture);

int mode

Specifies how arguments `colour`, `material` and `texture` are to be interpreted (see description below).

int colour

Index into palette colour table.

int material

Index into palette material table.

int texture

Index into palette texture table.

Description

This function creates and returns a new set of surface parameters, which when used (by **mav_surfaceParamsUse**) controls “what colour” is subsequent used to render objects in a window.

`mode` can take one of the following values:

- `MAV_COLOUR`
  to specify the use of an ambient colour
- `MAV_MATERIAL`
  to specify the use of a material type
- `MAV_TEXTURE`
  to specify the use of a decal texture
- `MAV_LIT_TEXTURE`
  to specify the use of a texture modulated by the material

- `MAV_BLENDED_TEXTURE`
  to specify the use of a blending between the material and texture depending on the texture's alpha value (0=material, 1=texture).

The other fields, `colour`, `material` and `texture`, respectively specify which colour, material and/or texture index to use from the palette associated with the window in which the object is being drawn. Only for the case of `MAV_LIT_TEXTURE` and `MAV_BLENDED_TEXTURE` does more than one index need to be given.

See also

MAV_surfaceParams, **mav_surfaceParamsUse**, **mav_paletteColourSet**, **mav_paletteMaterialSet**, **mav_paletteTextureSet**

mav_timerStart

mav_timerStart, mav_timerStop, mav_timeGet

Summary

Time and timer functions.

Syntax

void mav_timerStart(MAV_timer *t);

void mav_timerStop(MAV_timer *t);

MAV_time mav_timeGet(void);

Description

- **mav_timeGet**
  returns the CPU time used by the current program, and also the wallclock time.
- **mav_timerStart**
  records in `t` the the current elapsed CPU time and wallclock time.
- **mav_timerStop**
  records the current CPU and wallclock times in the `end` field of `t`, and computes the elapsed time, storing it in the `elapsed` field.

---

mav_vectorAdd

MAVERIK Level 1 functions

mav_vectorAdd, mav_vectorCrossProduct, mav_vectorSet, mav_vectorDotProduct, mav_vectorMult, mav_vectorMult3x3, mav_vectorMult4x4, mav_vectorNormalize, mav_vectorRotate, mav_vectorScalar, mav_vectorSub, mav_vectorMag

---

Summary

Vector manipulation utility functions.

Syntax

MAV_vector mav_vectorAdd(MAV_vector v1, MAV_vector v2);

MAV_vector mav_vectorCrossProduct(MAV_vector v1, MAV_vector v2);

MAV_vector mav_vectorSet(float x, float y, float z);

float mav_vectorDotProduct(MAV_vector v1, MAV_vector v2);

MAV_vector mav_vectorMult(MAV_vector v, MAV_matrix m);

MAV_vector mav_vectorMult3x3(MAV_vector v, MAV_matrix m);

MAV_vector mav_vectorMult4x4(MAV_vector v, MAV_matrix m);

MAV_vector mav_vectorNormalize(MAV_vector v);

MAV_vector mav_vectorRotate(MAV_vector v, MAV_vector ax, float ang);

MAV_vector mav_vectorScalar(MAV_vector v1, float f);

MAV_vector mav_vectorSub(MAV_vector v1, MAV_vector v2);

float mav_vectorMag(MAV_vector v1);

Description

- **mav_vectorAdd**
  adds the two 3D vectors `v1` and `v2`, returning the result vector.
- **mav_vectorCrossProduct**
  computes the cross product of the two 3D vectors `v1` and `v2`, and returns the resulting vector.
- **mav_vectorSet**
  creates the 3D column vector (`x`,`y`,`z`).
- **mav_vectorDotProduct**
  computes the dot product of the two 3D vectors `v1` and `v2` returning the result.

- **mav_vectorMult**
  multiplies the 3D vector `v` by the 4x3 matrix `m`, returning the multiplied vector.
- **mav_vectorMult3x3**
  multiplies the 3D vector `v` by the 3x3 matrix `m`, returning the multiplied vector.
- **mav_vectorMult4x4**
  multiplies the 3D vector `v` by the 4x4 matrix `m`, returning the multiplied vector.
- **mav_vectorNormalize**
  normalises the 3D vector (`x`,`y`,`z`), returning the normalised vector.
- **mav_vectorRotate**
  rotates the vector `v` by `ang` radians about an axis defined by `ax`, which passes through the origin. The resulting rotated vector is returned.
- **mav_vectorScalar**
  multiplies the 3D vector `v1` by `f`, and returns the resulting vector.
- **mav_vectorSub**
  computes (`v1-v2`), and returns the result vector.
- **mav_vectorMag**
  computes and returns the magnitude of the vector.

mav_vectorScrnPos

MAVERIK Level 1 functions

Summary

Compute screen coordinates of a 3D vector.

Syntax

MAV_vector mav_vectorScrnPos(MAV_vector v);

MAV_vector v

3D vector.

Description

This function computes the NDC screen coordinates (in the current window) of the 3D world coordinate frame vector `v`, i.e the position on screen the given vector maps to when it is transformed by the current projection matrix.

mav_vectorWorldPos

Summary

Compute world coordinates of a 3D vector.

Syntax

MAV_vector mav_vectorWorldPos(MAV_vector v);

MAV_vector v

3D vector.

Description

This function computes the world coordinates of the 3D NDC screen coordinate frame vector `v`, i.e. the world position vector which, when it is transformed by the current projection matrix, maps to the given NDC vector.

---

mav_viewParamsFixed

MAVERIK Level 1 functions

Summary

Set a fixed view point.

Syntax

void mav_viewParamsFixed(MAV_window *w);

MAV_window *w

window.

Description

When set as the `mod` field in a `MAV_viewParams` struct, this function specifies a “fixed” view point, so that the view parameters used are the same as those specified. Only advanced users would set `mod` to be anything other than this value. Setting `mod` to NULL is equivalent to setting it to this function.

For a full description of the role of the `mode` field, see the `MAV_viewParams` data structure.

---

mav_windowBackfaceCullGet

Summary

Get the backface culling status for a window.

Syntax

int mav_windowBackfaceCullGet(MAV_window *w);

MAV_window *w

Window.

Description

This function returns whether backface culling is enabled for a given window.

mav_windowBackfaceCullSet

MAVERIK Level 1 functions

Summary

Set backface culling for a window.

Syntax

void mav_windowBackfaceCullSet(MAV_window *w, int v);

MAV_window *w

Window.

int v

Set backface culling on or off.

Description

This function sets backface culling for a window to be on (v = `MAV_TRUE`) or off (v = `MAV_FALSE`).

mav_windowBackgroundColourSet

Summary

Set background colour of window.

Syntax

void mav_windowBackgroundColourSet(MAV_window *w, float r, float g, float b);

MAV_window *w

Window.

float r, float g, float b

Red, green and blue components of colour (0.0-1.0).

Description

This function sets the background colour of window `w` to be (`r`, `g`, `b`). If `w` is set to `mav_win_all`, this function will change the background colour of all windows. The default value of a window's background colour is (0.0, 0.5, 1.0).

---

mav_windowBlendSet

MAVERIK Level 1 functions

Summary

Set alpha blending for a window.

Syntax

void mav_windowBlendSet(MAV_window *w, int v);

MAV_window *w

Window.

int v

Set alpha blending on or off.

Description

This function sets alpha blending for a window to be on (v = `MAV_TRUE`) or off (v = `MAV_FALSE`).

mav_windowDelete

Summary

Delete a window.

Syntax

void mav_windowDelete(MAV_window *w);

Description

This function closes the window `w`. `w` should not be set `mav_win_all` or to the first window opened (you can't delete the very first window opened since this has a load of OpenGL state which other windows share). The functions **mav_windowNew** and **mav_windowDelete** are good for temporary pop up type windows.

---

mav_windowDump

MAVERIK Level 1 functions

Summary

Dump the contents of a window.

Syntax

void mav_windowDump(MAV_window *w, char *filename);

MAV_window *w

Window to dump.

char *filename

Name of output file for window dump.

Description

This function dumps the pixel contents of window `w` to file `filename`, in ppm format (RAW encoding).

---

mav_windowFogSet

Summary

Set the fog for a window.

Syntax

void mav_windowFogSet(MAV_window *w, int type, float data1, float data2, float r, float g, float b);

MAV_window *w

Window.

int type

The type of fog.

float data1, data2

Fog data (see below).

float r, g, b

Fog colour.

Description

This function sets the fog for window `w` to be the values supplied. `type` defines the type of fog used and can take the values `MAV_FOG_NONE` to disable fogging, `MAV_FOG_LINEAR` for a linear fog falloff and `MAV_FOG_EXP` and `MAV_FOG_EXP2` for exponential fog falloffs. For linear fog `data1` and `data2` define the start and end distances from the eye point where fog is applied. For exponential fog `data1` specifies the fog density and `data2` is unused. `r`, `g` and `b` define the colour of the fog. If the colour values are negative the background colour is used.

mav_windowLineStippleSet

MAVERIK Level 1 functions

Summary

Set the line stipple for a window.

Syntax

void mav_windowLineStippleSet(MAV_window *w, int factor, unsigned short pattern);

MAV_window *w

Window.

int factor

Stipple factor.

unsigned short pattern

Stipple pattern.

Description

This function sets the line stipple for window `w` to be the values supplied. Call with `factor` less than 1 to disable stippling.

mav_windowLineWidthSet MAVERIK Level 1 functions

Summary

Set the line width for a window.

Syntax

void mav_windowLineWidthSet(MAV_window *w, float wd);

MAV_window *w

Window.

float wd

Line width.

Description

This function sets the line width for window `w` to be `wd`.

mav_windowMultiSampleSet

MAVERIK Level 1 functions

Summary

Set multi-sampling for a window.

Syntax

void mav_windowMultiSampleSet(MAV_window *w, int v);

MAV_window *w

Window.

int v

Set multi-sampling on or off.

Description

This function sets the multi-sampling for window `w` to be on (v = `MAV_TRUE`) or off (v = `MAV_FALSE`). In order for this function to operate, the window must be capable of multi-sampling (i.e. `mav_opt_multiSample` must be set to `MAV_TRUE` before the window is opened).

mav_windowNew

Summary

Create a new window.

Syntax

MAV_window *mav_windowNew(int x, int y, int w, int h, char *name, char *disp);

int x

Left position of window in pixels.

int y

Top position of window in pixels.

int w

Width of window in pixels.

int h

Height of window in pixels.

char *name

Name of window.

char *disp

Name of X display on which to open window.

Description

This function creates a new window, returning a handle to it. The window is opened on the X display specified by `disp` (if set to `NULL`, then the display defined by the `DISPLAY` environment variable is used). The position of the top left corner of the window is given by (`x`, `y`), where the top left coordinates of the display screen are (0,0). The width and height of the window, in pixels, are respectively `w` and `h`. If the window manager permits, the title bar of the window will be labelled `name`.

mav_windowOrthogonalSet

MAVERIK Level 1 functions

Summary

Set orthogonal projection parameters for a window.

Syntax

void mav_windowOrthogonalSet(MAV_window *w, float ncp, float fcp, float size, float aspect);

MAV_window *w

Window.

float ncp

Position of near clip plane, in application units.

float fcp

Position of far clip plane, in application units.

float size

Vertical size, in application units, of the orthogonal projection

float aspect

Aspect ratio.

Description

This function sets the orthogonal projection parameters for window `w`. `ncp` and `fcp` respectively specify the positions of the near and far clip planes. `size` specifies the vertical extent of projection. `aspect` gives the aspect ratio of the view, which will normally be set to match the aspect ratio of window `w`.

---

mav_windowPerspectiveSet

Summary

Set perspective parameters for a window.

Syntax

**void mav_windowPerspectiveSet(MAV_window *w, float ncp, float fcp, float fov,
float aspect);**

MAV_window *w

Window.

float ncp

Position of near clip plane, in application units.

float fcp

Position of far clip plane, in application units.

float fov

Vertical field of view in degrees

float aspect

Aspect ratio.

Description

This function sets the perspective parameters for window `w`. `ncp` and `fcp` respectively specify the positions of the near and far clip planes. `fov` specifies the field of view, measured vertically in degrees `fov`/2 up and `fov2` down from the view direction vector. `aspect` gives the aspect ratio of the view, which will normally be set to match the aspect ratio of window `w`.

---

mav_windowPolygonModeSet

MAVERIK Level 1 functions

Summary

Set the polygon drawing mode for a window.

Syntax

void mav_windowPolygonModeSet(MAV_window *w, int v);

MAV_window *w

Window.

int v

Polygon mode.

Description

This function sets the polygon drawing mode for window `w`. Polygons may be drawn as vector outlines (`v` = `MAV_POLYGON_LINE`), or filled (`v` = `MAV_POLYGON_FILL`).

---

mav_windowSet

Summary

Set window to be active window for rendering.

Syntax

void mav_windowSet(MAV_window *w);

MAV_window *w

Window.

Description

This function sets `w` to be the active window for rendering.

---

mav_windowViewModifierSet

MAVERIK Level 1 functions

Summary

Set the window view modification function and parameters.

Syntax

void mav_windowViewModifierSet(MAV_window *w, MAV_viewModifierParams *vmp, MAV_viewModifierFn fn);

MAV_window *w

Window.

MAV_viewModifierParams *vmp

The view modification parameters to use.

MAV_viewModifierFn fn

The view modification function to use.

Description

This function sets the view modification function for window `w` to be `fn` and to use an associated set of parameters `vmp`. The function arbitrarily modifiers the view in a window by an amount defined by the associated parameters.

This function is typically used to perform stereo viewing as follows. Two windows will share a common set of view modification parameters, but one window will have a view modification function which translates the view to the right by an amount given in the view modification parameters, while the other window will have a different view modification function which translates the view to the left by the same amount.

MAVERIK supplies three view modification functions: **mav_eyeMono**, **mav_eyeLeft** and **mav_eyeRight**. **mav_eyeMono** does not modify the view, and is the default for a window, while **mav_eyeLeft** and **mav_eyeRight** respectively offset the eye point to the left and right along the view right vector by half the value of the `offset` set in the `MAV_viewModifierParams`. An advanced user could write their own `MAV_viewModifierFn`'s so as to apply, for example, a convergence in the view direction vectors as well as an eye point offset.

```
MAV_viewModiferParams vmp;
```

```
vmp.offset= 20.0;

mav_windowViewModifierSet(mav_win_left, &vmp, mav_eyeLeft);
mav_windowViewModifierSet(mav_win_right, &vmp, mav_eyeRight);
```

---

mav_windowViewParamsSet

MAVERIK Level 1 functions

Summary

Set the view parameters for a window.

Syntax

void mav_windowViewParamsSet(MAV_window *w, MAV_viewParams *vp);

MAV_window *w

Window.

MAV_viewParams *vp

View parameters.

Description

This function sets the view parameters for window `w` to be `vp`.

Chapter 6

Level 2 functions

mav_BBCullToClipPlanes

MAVERIK Level 2 functions

mav_BBCullToClipPlanes, mav_BBGetCorner, mav_BBIntersectsLine, mav_BBIntersectsBB, mav_BBInsideBB, mav_BBIntersectsClipPlanes, mav_BBAlign, mav_BBCompInit, mav_BBCompBB, mav_BBCompPt

Summary

Bounding box utility functions.

Syntax

int mav_BBCullToClipPlanes(MAV_BB bb, MAV_clipPlanes cp);

int mav_BBGetCorner(MAV_vector v);

int mav_BBIntersectsLine(MAV_BB bb, MAV_line ln, MAV_objectIntersection *oi);

int mav_BBIntersectsBB(MAV_BB bb1, MAV_BB bb2);

int mav_BBInsideBB(MAV_BB bb1, MAV_BB bb2);

int mav_BBIntersectsClipPlanes(MAV_BB bb, int *clist, MAV_clipPlanes *cp);

void mav_BBAlign(MAV_BB in, MAV_matrix m, MAV_BB *out);

void mav_BBCompInit(MAV_BB *bb);

void mav_BBCompBB(MAV_BB in, MAV_BB *out);

void mav_BBCompPt(MAV_vector v, MAV_BB *bb);

Description

- **mav_BBCullToClipPlanes**
  like **mav_BBCull** but uses the clip planes given rather than those which correspond to the current viewing frustum.
- **mav_BBGetCorner**
  takes a vector `v` defining the normal to a plane, and returns an integer representing which corners of an axis-aligned box need to be tested when testing for intersection against that plane. This means that only 2 corners are tested for each box, instead of 8.
- **mav_BBIntersectsLine**
  intersects bounding box `bb` with line `ln` and returns the closest intersection point (if any) in `oi`. The result of the function is MAV_TRUE if an intersection was found, else MAV_FALSE.
- **mav_BBIntersectsBB**
  tests for intersects between bounding boxes `bb1` and `bb2`. The result of the function is MAV_TRUE if they intersect, else MAV_FALSE.
- **mav_BBInsideBB**
  tests for both permutations of complete enclosure between bounding boxes `bb1` and `bb2`. The result of the function is MAV_TRUE if one is inside the other, else MAV_FALSE.
- **mav_BBIntersectsClipPlanes**
  intersects an axis-aligned box `bb` with a set of clip planes `cp`. A return value of 0 indicates no intersection; of 1 indicates the box is entirely inside clip planes, 2 = box crosses clip planes.
- **mav_BBAlign**
  takes the axis-aligned bounding box `in`, transforms it with matrix `m`, and then transforms it again to align it with the world coordinate axes.

The following functions are for computing the bounding box of a composite object.

- **mav_BBCompInit**
  initialises the bounding box `bb`.
- **mav_BBCompBB**
  calculates the enclosing bounding box of two separate bounding boxes, `in` and `out`. The resulting bounding box overwrites `out`.
- **mav_BBCompPt**
  calculates the enclosing bounding box of bounding box `bb` and point `v`. The resulting bounding box overwrites `bb`.

mav_SMSDisplayFn

Summary

The default SMS display function.

Syntax

void mav_SMSDisplayFn(MAV_object *o, MAV_drawInfo *di, void *ud);

MAV_object *o

Object.

MAV_drawInfo *di

Drawing information.

void *ud

User defined data.

Description

When an SMS is displayed with **mav_SMSDisplay** (or its variants), it is the function defined by the variable **mav_SMS_displayFn** which gets called in order to display each object.

The function **mav_SMSDisplayFn** is the default value of **mav_SMS_displayFn** and simply executes the draw callback of the object in order to display it.

mav_SMSDisplayUsingDrawInfo

MAVERIK Level 2 functions

Summary

.

Syntax

int mav_SMSDisplayUsingDrawInfo(MAV_window *w, MAV_SMS *s, MAV_drawInfo di);

Description

mav_SMSDisplayUsingDrawInfo allows an application to specify its own drawing information (in `di`), instead of deriving these from the current viewing parameters, as is normally the case. This is necessary, for example, if an application wishing the culling frustum to be different from the viewing frustum.

The return value of this function is MAV_TRUE or MAV_FALSE and respectively indicates the success or failure of this operation.

mav_SMSIntersectBBAll

Summary

Intersect all SMSs with a BB.

Syntax

int mav_SMSIntersectBBAll(MAV_window *w, MAV_BB bb, MAV_SMS *objs);

MAV_window *w

Window.

MAV_BB bb

BB for intersection test.

MAV_SMS *objs

The intersected objects (if any).

Description

The function intersects all objects in all SMSs with BB `bb`. If an intersection is detected, the function returns MAV_TRUE, otherwise MAV_FALSE. Object which intersect the BB are added to SMS `objs`.

mav_SMSIntersectBB

MAVERIK Level 2 functions

Summary

Intersect an SMS with a BB.

Syntax

int mav_SMSIntersectBB(MAV_window *w, MAV_SMS *sms, MAV_BB bb, MAV_SMS *objs);

MAV_window *w

Window.

MAV_SMS *sms

The SMS.

MAV_BB bb

BB for intersection test.

MAV_SMS *objs

The intersected objects (if any).

Description

The function intersects all objects in the given SMS with BB `bb`. If an intersection is detected, the function returns MAV_TRUE, otherwise MAV_FALSE. Object which intersect the BB are added to SMS `objs`.

mav_SMSIntersectLineAll

Summary

Intersect all SMSs with a line.

Syntax

int mav_SMSIntersectLineAll(MAV_window *w, MAV_line ln, MAV_objectIntersection *oi, MAV_object **o);

MAV_window *w

Window.

MAV_line ln

Line for intersection test.

MAV_objectIntersection *oi

Intersection point (if any)

MAV_object **o

Intersection object (if any)

Description

The function intersects all objects in all SMSs with line `ln`. If an intersection is detected, the function returns MAV_TRUE, otherwise MAV_FALSE. The closest intersection point is returned in `oi`, and a pointer to the intersecting object in `o`.

mav_SMSIntersectLine

MAVERIK Level 2 functions

Summary

Intersect an SMS with a line.

Syntax

**int mav_SMSIntersectLine(MAV_window *w, MAV_SMS *sms, MAV_line ln,
MAV_objectIntersection *oi, MAV_object **o);**

MAV_window *w

The window.

MAV_SMS *sms

The SMS.

MAV_line ln

Line for intersection test.

MAV_objectIntersection *oi

Intersection point (if any)

MAV_object **o

Intersection object (if any)

Description

The function intersects all objects in a given SMS with line `ln`. If an intersection is detected, the function returns MAV_TRUE, otherwise MAV_FALSE. The closest intersection point is returned in `oi`, and a pointer to the intersecting object in `o`.

mav_boxBB

MAVERIK Level 2 functions

mav_boxBB, mav_compositeBB, mav_coneBB, mav_ctorusBB, mav_cylinderBB, mav_ellipseBB, mav_facetBB, mav_hellipseBB, mav_hsphereBB, mav_polylineBB, mav_polygonBB, mav_polygonGrpBB, mav_pyramidBB, mav_rectangleBB, mav_rtorusBB, mav_SMSObjBB, mav_sphereBB, mav_teapotBB, mav_textBB, mav_avatarBB, mav_TDMCursorBB

Summary

Compute axis-aligned bounding box for Maverik default object classes.

Syntax

int mav_boxBB(MAV_object *o, MAV_BB *bb);

int mav_compositeBB(MAV_object *o, MAV_BB *bb);

int mav_coneBB(MAV_object *o, MAV_BB *bb);

int mav_ctorusBB(MAV_object *o, MAV_BB *bb);

int mav_cylinderBB(MAV_object *o, MAV_BB *bb);

int mav_ellipseBB(MAV_object *o, MAV_BB *bb);

int mav_facetBB(MAV_object *o, MAV_BB *bb);

int mav_hellipseBB(MAV_object *o, MAV_BB *bb);

int mav_hsphereBB(MAV_object *o, MAV_BB *bb);

int mav_polylineBB(MAV_object *o, MAV_BB *bb);

int mav_polygonBB(MAV_object *o, MAV_BB *bb);

int mav_polygonGrpBB(MAV_object *o, MAV_BB *bb);

int mav_pyramidBB(MAV_object *o, MAV_BB *bb);

int mav_rectangleBB(MAV_object *o, MAV_BB *bb);

int mav_rtorusBB(MAV_object *o, MAV_BB *bb);

int mav_SMSObjBB(MAV_object *o, MAV_BB *bb);

int mav_sphereBB(MAV_object *o, MAV_BB *bb);

int mav_teapotBB(MAV_object *o, MAV_BB *bb);

int mav_textBB(MAV_object *o, MAV_BB *bb);

int mav_avatarBB(MAV_object *o, MAV_BB *bb);

int mav_TDMCursorBB(MAV_object *o, MAV_BB *bb);

Description

mav_boxBB (and the corresponding functions for other object classes) computes the axis-aligned bounding box for object `o`, returning the result in `bb`. **mav_boxBB** is optimised for speed at the expense of accuracy, and computes a box which may be an over-estimate of the true dimensions of the bounding box. This function is registered by **mav_initialise** as the default bounding box callback for each object class.

A corresponding set of bounding box computation functions named **mav_boxBB2** (and similarly for other object classes) is also available, for applications which require an accurate (but necessarily slower) computation of the bounding box.

Explain BB2

---

mav_boxDraw

MAVERIK Level 2 functions

mav_boxDraw, mav_compositeDraw, mav_coneDraw, mav_ctorusDraw, mav_cylinderDraw, mav_ellipseDraw, mav_facetDraw, mav_hellipseDraw, mav_hsphereDraw, mav_polylineDraw, mav_polygonDraw, mav_polygonGrpDraw, mav_pyramidDraw, mav_rectangleDraw, mav_rtorusDraw, mav_SMSObjDraw, mav_sphereDraw, mav_teapotDraw, mav_textDraw, mav_avatarDraw, mav_TDMCursorDraw

---

Summary

Rendering callback for Maverik default object classes.

Syntax

int mav_boxDraw(MAV_object *o, MAV_drawInfo *di);

int mav_compositeDraw(MAV_object *o, MAV_drawInfo *di);

int mav_coneDraw(MAV_object *o, MAV_drawInfo *di);

int mav_ctorusDraw(MAV_object *o, MAV_drawInfo *di);

int mav_cylinderDraw(MAV_object *o, MAV_drawInfo *di);

int mav_ellipseDraw(MAV_object *o, MAV_drawInfo *di);

int mav_facetDraw(MAV_object *o, MAV_drawInfo *di);

int mav_hellipseDraw(MAV_object *o, MAV_drawInfo *di);

int mav_hsphereDraw(MAV_object *o, MAV_drawInfo *di);

int mav_polylineDraw(MAV_object *o, MAV_drawInfo *di);

int mav_polygonDraw(MAV_object *o, MAV_drawInfo *di);

int mav_polygonGrpDraw(MAV_object *o, MAV_drawInfo *di);

int mav_pyramidDraw(MAV_object *o, MAV_drawInfo *di);

int mav_rectangleDraw(MAV_object *o, MAV_drawInfo *di);

int mav_rtorusDraw(MAV_object *o, MAV_drawInfo *di);

int mav_SMSObjDraw(MAV_object *o, MAV_drawInfo *di);

int mav_sphereDraw(MAV_object *o, MAV_drawInfo *di);

int mav_teapotDraw(MAV_object *o, MAV_drawInfo *di);

int mav_textDraw(MAV_object *o, MAV_drawInfo *di);

int mav_avatarDraw(MAV_object *o, MAV_drawInfo *di);

int mav_TDMCursorDraw(MAV_object *o, MAV_drawInfo *di);

MAV_object *o

Object to draw.

MAV_drawInfo *di

Drawing information.

Description

This is a callback function, which draws an object. It is given as the argument to **mav_callbackDrawSet** for each class of object defined in the common objects module. `o` is the object to draw, and `di` is drawing information including view parameters and culling information.

See also

mav_callbackDrawSet (page 260).

mav_boxDump

mav_boxDump, mav_compositeDump, mav_coneDump, mav_ctorusDump, mav_cylinderDump, mav_ellipseDump, mav_facetDump, mav_hellipseDump, mav_hsphereDump, mav_polylineDump, mav_polygonDump, mav_polygonGrpDump, mav_pyramidDump, mav_rectangleDump, mav_rtorusDump, mav_SMSObjDump, mav_sphereDump, mav_teapotDump, mav_textDump, mav_avatarDump, mav_TDMCursorDump

Summary

Print data of object.

Syntax

int mav_boxDump(MAV_object *o);

int mav_compositeDump(MAV_object *o);

int mav_coneDump(MAV_object *o);

int mav_ctorusDump(MAV_object *o);

int mav_cylinderDump(MAV_object *o);

int mav_ellipseDump(MAV_object *o);

int mav_facetDump(MAV_object *o);

int mav_hellipseDump(MAV_object *o);

int mav_hsphereDump(MAV_object *o);

int mav_polylineDump(MAV_object *o);

int mav_polygonDump(MAV_object *o);

int mav_polygonGrpDump(MAV_object *o);

int mav_pyramidDump(MAV_object *o);

int mav_rectangleDump(MAV_object *o);

int mav_rtorusDump(MAV_object *o);

int mav_SMSObjDump(MAV_object *o);

int mav_sphereDump(MAV_object *o);

int mav_teapotDump(MAV_object *o);

int mav_textDump(MAV_object *o);

int mav_avatarDump(MAV_object *o);

int mav_TDMCursorDump(MAV_object *o);

MAV_object *o

Object to query.

Description

This function prints to `stdout` the data in object `o`.

---

mav_boxGetMatrix

mav_boxGetMatrix, mav_compositeGetMatrix, mav_coneGetMatrix, mav_ctorusGetMatrix, mav_cylinderGetMatrix, mav_ellipseGetMatrix, mav_facetGetMatrix, mav_hellipseGetMatrix, mav_hsphereGetMatrix, mav_polylineGetMatrix, mav_polygonGetMatrix, mav_polygonGrpGetMatrix, mav_pyramidGetMatrix, mav_rectangleGetMatrix, mav_rtorusGetMatrix, mav_SMSObjGetMatrix, mav_sphereGetMatrix, mav_teapotGetMatrix, mav_textGetMatrix, mav_avatarGetMatrix

Summary

Get matrix of object.

Syntax

int mav_boxGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_compositeGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_coneGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_ctorusGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_cylinderGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_ellipseGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_facetGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_hellipseGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_hsphereGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_polylineGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_polygonGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_polygonGrpGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_pyramidGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_rectangleGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_rtorusGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_SMSObjGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_sphereGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_teapotGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_textGetMatrix(MAV_object *o, MAV_matrix **m);

int mav_avatarGetMatrix(MAV_object *o, MAV_matrix **m);

MAV_object *o

Object to query.

MAV_matrix **m

Pointer to object's matrix

Description

This function returns a pointer to the matrix of object `o`.

---

mav_boxGetSurfaceParams

mav_boxGetSurfaceParams, mav_compositeGetSurfaceParams, mav_coneGetSurfaceParams, mav_ctorusGetSurfaceParams, mav_cylinderGetSurfaceParams, mav_ellipseGetSurfaceParams, mav_facetGetSurfaceParams, mav_hellipseGetSurfaceParams, mav_hsphereGetSurfaceParams, mav_polylineGetSurfaceParams, mav_polygonGetSurfaceParams, mav_polygonGrpGetSurfaceParams, mav_pyramidGetSurfaceParams, mav_rectangleGetSurfaceParams, mav_rtorusGetSurfaceParams, mav_sphereGetSurfaceParams, mav_teapotGetSurfaceParams, mav_textGetSurfaceParams, mav_avatarGetSurfaceParams, mav_SMSObjGetSurfaceParams, mav_TDMCursorGetSurfaceParams

---

Summary

Get surface parameters of object.

Syntax

int mav_boxGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_compositeGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_coneGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_ctorusGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_cylinderGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_ellipseGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_facetGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_hellipseGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_hsphereGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_polylineGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_polygonGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_polygonGrpGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_pyramidGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_rectangleGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_rtorusGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_sphereGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_teapotGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_textGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_avatarGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_SMSObjGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

int mav_TDMCursorGetSurfaceParams(MAV_object *o, MAV_surfaceParams ***sp);

MAV_object *o

Object to query.

MAV_surfaceParams ***sp

Pointer to object's surface parameters.

Description

This function returns a pointer to the surface parameters of object `o`.

---

mav_boxGetUserdef

mav_boxGetUserdef, mav_compositeGetUserdef, mav_coneGetUserdef, mav_ctorusGetUserdef, mav_cylinderGetUserdef, mav_ellipseGetUserdef, mav_facetGetUserdef, mav_hellipseGetUserdef, mav_hsphereGetUserdef, mav_polylineGetUserdef, mav_polygonGetUserdef, mav_polygonGrpGetUserdef, mav_pyramidGetUserdef, mav_rectangleGetUserdef, mav_rtorusGetUserdef, mav_SMSObjGetUserdef, mav_sphereGetUserdef, mav_teapotGetUserdef, mav_textGetUserdef, mav_avatarGetUserdef, mav_TDMCursorGetUserdef

Summary

Get user-defined data of object.

Syntax

int mav_boxGetUserdef(MAV_object *o, void ***ud);

int mav_compositeGetUserdef(MAV_object *o, void ***ud);

int mav_coneGetUserdef(MAV_object *o, void ***ud);

int mav_ctorusGetUserdef(MAV_object *o, void ***ud);

int mav_cylinderGetUserdef(MAV_object *o, void ***ud);

int mav_ellipseGetUserdef(MAV_object *o, void ***ud);

int mav_facetGetUserdef(MAV_object *o, void ***ud);

int mav_hellipseGetUserdef(MAV_object *o, void ***ud);

int mav_hsphereGetUserdef(MAV_object *o, void ***ud);

int mav_polylineGetUserdef(MAV_object *o, void ***ud);

int mav_polygonGetUserdef(MAV_object *o, void ***ud);

int mav_polygonGrpGetUserdef(MAV_object *o, void ***ud);

int mav_pyramidGetUserdef(MAV_object *o, void ***ud);

int mav_rectangleGetUserdef(MAV_object *o, void ***ud);

int mav_rtorusGetUserdef(MAV_object *o, void ***ud);

int mav_SMSObjGetUserdef(MAV_object *o, void ***ud);

int mav_sphereGetUserdef(MAV_object *o, void ***ud);

int mav_teapotGetUserdef(MAV_object *o, void ***ud);

int mav_textGetUserdef(MAV_object *o, void ***ud);

int mav_avatarGetUserdef(MAV_object *o, void ***ud);

int mav_TDMCursorGetUserdef(MAV_object *o, void ***ud);

MAV_object *o

Object to query.

void ***ud

Pointer to the ”user-defined data”.

Description

This function returns in `ud` a pointer to the “user-defined data” field of object `a`.

---

mav_boxID

mav_boxID, mav_compositeID, mav_coneID, mav_ctorusID, mav_cylinderID, mav_ellipseID, mav_facetID, mav_hellipseID, mav_hsphereID, mav_polylineID, mav_polygonID, mav_polygonGrpID, mav_pyramidID, mav_rectangleID, mav_rtorusID, mav_SMSObjID, mav_sphereID, mav_teapotID, mav_textID, mav_avatarID, mav_TDMCursorID

Summary

Get identifier for object class.

Syntax

int mav_boxID(MAV_object *o, char **id);

int mav_compositeID(MAV_object *o, char **id);

int mav_coneID(MAV_object *o, char **id);

int mav_ctorusID(MAV_object *o, char **id);

int mav_cylinderID(MAV_object *o, char **id);

int mav_ellipseID(MAV_object *o, char **id);

int mav_facetID(MAV_object *o, char **id);

int mav_hellipseID(MAV_object *o, char **id);

int mav_hsphereID(MAV_object *o, char **id);

int mav_polylineID(MAV_object *o, char **id);

int mav_polygonID(MAV_object *o, char **id);

int mav_polygonGrpID(MAV_object *o, char **id);

int mav_pyramidID(MAV_object *o, char **id);

int mav_rectangleID(MAV_object *o, char **id);

int mav_rtorusID(MAV_object *o, char **id);

int mav_SMSObjID(MAV_object *o, char **id);

int mav_sphereID(MAV_object *o, char **id);

int mav_teapotID(MAV_object *o, char **id);

int mav_textID(MAV_object *o, char **id);

int mav_avatarID(MAV_object *o, char **id);

int mav_TDMCursorID(MAV_object *o, char **id);

MAV_object *o

Object to query.

char **id

Identifier string returned.

Description

This function returns in `id` an identifier string for object class `o`. For example, a sphere object might return “sphere” for sphere.

---

mav_boxIntersect

mav_boxIntersect, mav_compositeIntersect, mav_coneIntersect, mav_ctorusIntersect, mav_cylinderIntersect, mav_ellipseIntersect, mav_facetIntersect, mav_hellipseIntersect, mav_hsphereIntersect, mav_polygonIntersect, mav_polygonGrpIntersect, mav_pyramidIntersect, mav_rectangleIntersect, mav_rtorusIntersect, mav_SMSObjIntersect, mav_sphereIntersect, mav_avatarIntersect

---

Summary

Calculate intersection of line with Maverik default object classes.

Syntax

int mav_boxIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_compositeIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_coneIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_ctorusIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_cylinderIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_ellipseIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_facetIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_hellipseIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_hsphereIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_polygonIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_polygonGrpIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_pyramidIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_rectangleIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_rtorusIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_SMSObjIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_sphereIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

int mav_avatarIntersect(MAV_object *o, MAV_line *ln, MAV_objectIntersection *oi);

MAV_object *o

Object to test for intersection.

MAV_line *ln

Line to test for intersection.

MAV_objectIntersection *oi

Intersection of line and object (if any).

Description

This function tests for intersection beween object `o` and line `ln`. If one or more intersections are detected, the closest intersection point is returned in `oi`, and the result of the function is MAV_TRUE. If no intersection is detected, the result of the function is MAV_FALSE.

If this callback function is not defined for object `o`, the intersection function will test against the object's bounding box.

mav_callbackDeleteExec

Summary

Execute object delete callback.

Syntax

int mav_callbackDeleteExec(MAV_window *w, MAV_object *o);

MAV_window *w

Window.

MAV_object *o

Object class.

Description

This function executes the "delete" callback for object `o`. The result of the callback is returned.

See also

mav_callbackDeleteSet (page 258).

mav_callbackDeleteSet

MAVERIK Level 2 functions

Summary

Set object delete callback.

Syntax

void mav_callbackDeleteSet(MAV_window *w, MAV_class *c, MAV_callbackDeleteFn fn);

MAV_window *w

Window.

MAV_class *c

Object class.

MAV_callbackDeleteFn fn

Callback function to be called on object deletion.

Description

This function sets the “delete” callback function for a given class `c` of object in window `w` to `fn`. The callback will be called on deletion of an object of the given class. The callback function has the opportunity to free any memory associated with the object’s data.

See also

mav_callbackDeleteExec (page 257).

mav_callbackDrawExec

mav_callbackDrawExec, mav_callbackBBExec, mav_callbackIntersectExec, mav_callbackIDExec, mav_callbackDumpExec, mav_callbackGetUserdefExec, mav_callbackGetMatrixExec, mav_callbackGetSurfaceParamsExec

Summary

Execute callback function for an object.

Syntax

int mav_callbackDrawExec(MAV_window *w, MAV_object *o, MAV_drawInfo *di);

int mav_callbackBBExec(MAV_window *w, MAV_object *o, MAV_BB *bb);

int mav_callbackIntersectExec(MAV_window *w, MAV_object *o, MAV_line ln, MAV_objectIntersection *oi);

int mav_callbackIDExec(MAV_window *w, MAV_object *o, char **id);

int mav_callbackDumpExec(MAV_window *w, MAV_object *o);

int mav_callbackGetUserdefExec(MAV_window *w, MAV_object *o, void ***ud);

int mav_callbackGetMatrixExec(MAV_window *w, MAV_object *o, MAV_matrix **m);

int mav_callbackGetSurfaceParamsExec(MAV_window *w, MAV_object *o, MAV_surfaceParams ***sp);

Description

These functions execute the appropriate callback function for object `o` in window `w`.

---

mav_callbackDrawSet

MAVERIK Level 2 functions

mav_callbackDrawSet, mav_callbackBBSet, mav_callbackIntersectSet, mav_callbackIDSet, mav_callbackDumpSet, mav_callbackGetUserdefSet, mav_callbackGetMatrixSet, mav_callbackGetSurfaceParamsSet

---

Summary

Set callback function for an object class.

Syntax

void mav_callbackDrawSet(MAV_window *w, MAV_class *c, MAV_callbackDrawFn fn);

void mav_callbackBBSet(MAV_window *w, MAV_class *c, MAV_callbackBBFn fn);

void mav_callbackIntersectSet(MAV_window *w, MAV_class *c, MAV_callbackIntersectFn fn);

void mav_callbackIDSet(MAV_window *w, MAV_class *c, MAV_callbackIDFn fn);

void mav_callbackDumpSet(MAV_window *w, MAV_class *c, MAV_callbackDumpFn fn);

void mav_callbackGetUserdefSet(MAV_window *w, MAV_class *c, MAV_callbackGetUserdefFn fn);

void mav_callbackGetMatrixSet(MAV_window *w, MAV_class *c, MAV_callbackGetMatrixFn fn);

void mav_callbackGetSurfaceParamsSet(MAV_window *w, MAV_class *c, MAV_callbackGetSurfaceParamsFn fn);

Description

These functions set the appropriate callback function for object class `c` in window `w` to be `fn`.

---

mav_callbackResizeSet

mav_callbackResizeSet, mav_callbackResizeExec, mav_resizeDefault, mav_callbackMapSet, mav_callbackMapExec, mav_mapDefault, mav_callbackCrossingSet, mav_callbackCrossingExec, mav_callbackExposeSet, mav_callbackExposeExec, mav_exposeDefault

---

Summary

Window management callback functions.

Syntax

void mav_callbackResizeSet(MAV_window *w, MAV_callbackResizeFn fn);

int mav_callbackResizeExec(MAV_window *w, MAV_resizeEvent *re);

int mav_resizeDefault(MAV_object *o, MAV_resizeEvent *re);

void mav_callbackMapSet(MAV_window *w, MAV_callbackMapFn fn);

int mav_callbackMapExec(MAV_window *w, MAV_mapEvent *me);

int mav_mapDefault(MAV_object *o, MAV_mapEvent *me);

void mav_callbackCrossingSet(MAV_window *w, MAV_callbackCrossingFn fn);

int mav_callbackCrossingExec(MAV_window *w, MAV_crossingEvent *ce);

void mav_callbackExposeSet(MAV_window *w, MAV_callbackExposeFn fn);

int mav_callbackExposeExec(MAV_window *w, MAV_exposeEvent *ee);

int mav_exposeDefault(MAV_object *o, MAV_exposeEvent *ee);

Description

These are the callback functions which respond to window events, as follows:

- Window resizing:
  mav_callbackResizeSet sets the callback function to be executed when window `w` is resized to be `fn`. **mav_callbackResizeExec** executes the callback function, passing it the resize event data `re`. **mav_resizeDefault** is the default resize callback registered by the system for a window when it is created.

- Window mapping (iconizing/restoring):
  mav_callbackMapSet sets the callback function to be executed when window `w` is mapped, to

be `fn`. **mav_callbackMapExec** executes the callback function, passing it the map event data `me`. **mav_mapDefault** is the default resize callback registered by the system for a window when it is created.

- Window crossing:
  mav_callbackCrossingSet sets the callback function to be executed when the mouse enters or leaves window `w` to be `fn`. **mav_callbackCrossingExec** executes the callback function, passing it the crossing event data `ce`.

- Window exposure:
  mav_callbackExposeSet sets the callback function to be executed when window `w` is exposed to be `fn`. **mav_callbackExposeExec** executes the callback function, passing it the expose event data `ee`. **mav_exposeDefault** is the default expose callback registered by the system for a window when it is created.

---

mav_classNew

Summary

Create a new class.

Syntax

MAV_class *mav_classNew(void);

Description

Creates a new class, returning a pointer to it.

---

mav_clipPlanesGet

MAVERIK Level 2 functions

mav_clipPlanesGet, mav_clipPlanesGetFromPixels, mav_clipPlanesGetFromBB

Summary

Obtain clip frames for 3D volume.

Syntax

MAV_clipPlanes mav_clipPlanesGet(MAV_window *w, float xmin, float xmax, float ymin, float ymax, float zmin, float zmax);

MAV_clipPlanes mav_clipPlanesGetFromPixels(MAV_window *w, int xmin, int xmax, int ymin, int ymax, float zmin, float zmax);

MAV_clipPlanes mav_clipPlanesGetFromBB(MAV_BB bb);

Description

mav_clipPlanesGet returns the clip planes corresponding to the 3D volume specified in NDC for window `w` by (`xmin`, `ymin`, `zmin`, `xmax`, `ymax`, `zmax`).

mav_clipPlanesGetFromPixels is the same, but the 3D volume is specified in pixels.

mav_clipPlanesGetFromBB generates a set of clip planes from a BB.

---

mav_drawInfoTransFrame

Summary

Transform drawing information to object coordinates.

Syntax

MAV_drawInfo mav_drawInfoTransFrame(MAV_drawInfo in, MAV_matrix mat);

MAV_drawInfo in

Drawing information.

MAV_matrix mat

Matrix.

Description

This function is analgous to **mav_lineTransFrame**, but transforms drawing information by the inverse of matrix `mat`.

See also

mav_lineTransFrame (page 273).

---

mav_gfx

MAVERIK Level 2 functions

mav_gfxClipPlaneSet, mav_gfxClipPlanesSet, mav_gfxClipPlaneEnable, mav_gfxClipPlaneDisable, mav_gfxClearC, mav_gfxClearZ, mav_gfxClearA, mav_gfxClearCZ, mav_gfxBackgroundColourSet, mav_gfxDepthTestSet, mav_gfxDepthMaskSet, mav_gfxNormalizeSet, mav_gfxBackfaceCullSet, mav_gfxBackfaceCullGet, mav_gfxBufferReadSet, mav_gfxPixelRead, mav_gfxPixelReadUByte, mav_gfxPixelDraw, mav_gfxViewPortSet, mav_gfxRasterPosSet, mav_gfxRasterPos2DSet, mav_gfxLineWidthSet, mav_gfxLineWidthGet, mav_gfxLineStippleSet, mav_gfxFogSet, mav_gfxVisualInfoGet, mav_gfxPolygonModeSet, mav_gfxMultiSampleSet, mav_gfxFinish, mav_gfxFlush, mav_gfxMatrixMode, mav_gfxMatrixLoad, mav_gfxMatrixPush, mav_gfxMatrixPop, mav_gfxMatrixMult, mav_gfxMatrixTranslate, mav_gfxMatrixScale, mav_gfxPerspectiveSet, mav_gfxOrthogonalSet, mav_gfxMatrixGet, mav_gfxPolygonBegin, mav_gfxPolygonEnd, mav_gfxTrianglesBegin, mav_gfxTrianglesEnd, mav_gfxStripQBegin, mav_gfxStripQEnd, mav_gfxStripTBegin, mav_gfxStripTEnd, mav_gfxLineClosedBegin, mav_gfxLineClosedEnd, mav_gfxLineBegin, mav_gfxLineEnd, mav_gfxMeshTBegin, mav_gfxMeshTEnd, mav_gfxVertex, mav_gfxNormal, mav_gfxTexCoord, mav_gfxColouringModeUse, mav_gfxColourSet, mav_gfxColourUse, mav_gfxMaterialSet, mav_gfxMaterialUse, mav_gfxTextureSet, mav_gfxTextureUse, mav_gfxLightingModelSet, mav_gfxLightingModelUse, mav_gfxLightSet, mav_gfxLightUse, mav_gfxLightPos, mav_gfxBlendSet, mav_gfxTextureEnv1Set, mav_gfxTextureEnv2Set, mav_gfxAccumSet, mav_gfxListsNew, mav_gfxListNew, mav_gfxListEnd, mav_gfxListExec, mav_gfxListsExec, mav_gfxListsDelete, mav_gfx3DfxModeSet, mav_gfx3DfxBoardSet

---

Summary

Wrapper functions to graphics system.

Syntax

void mav_gfxClipPlaneSet(int id, MAV_clipPlane cp);

void mav_gfxClipPlanesSet(MAV_clipPlanes *cp);

void mav_gfxClipPlaneEnable(int id);

void mav_gfxClipPlaneDisable(int id);

void mav_gfxClearC(void);

void mav_gfxClearZ(void);

```
void mav_gfxClearA(void);

void mav_gfxClearCZ(void);

void mav_gfxBackgroundColourSet(float r, float g, float b);

void mav_gfxDepthTestSet(int v);

void mav_gfxDepthMaskSet(int v);

void mav_gfxNormalizeSet(int v);

void mav_gfxBackfaceCullSet(int v);

int mav_gfxBackfaceCullGet(void);

void mav_gfxBufferReadSet(int buf);

void mav_gfxPixelRead(int x, int y, int w, int h, unsigned long *d);

void mav_gfxPixelReadUByte(int x, int y, int w, int h, unsigned char *d);

void mav_gfxPixelDraw(int w, int h, float *v);

void mav_gfxViewPortSet(int x, int y, int w, int h);

void mav_gfxRasterPosSet(MAV_vector v);

void mav_gfxRasterPos2DSet(float x, float y);

void mav_gfxLineWidthSet(float wd);

float mav_gfxLineWidthGet(void);

void mav_gfxLineStippleSet(int factor, unsigned short pattern);

void mav_gfxFogSet(int type, float data1, float data2, float r, float g, float b);

int mav_gfxVisualInfoGet(int *r, int *g, int *b, int *a, int *d, int *db, int *ar,
    int *ag, int *ab, int *aa, int *sb, int *msb);

void mav_gfxPolygonModeSet(int v);

void mav_gfxMultiSampleSet(int v);

void mav_gfxFinish(void);

void mav_gfxFlush(void);

void mav_gfxMatrixMode(int mode);

void mav_gfxMatrixLoad(MAV_matrix m);

void mav_gfxMatrixPush(void);

void mav_gfxMatrixPop(void);

void mav_gfxMatrixMult(MAV_matrix m);

void mav_gfxMatrixTranslate(MAV_vector v);

void mav_gfxMatrixScale(float x, float y, float z);

void mav_gfxPerspectiveSet(float ncp, float fcp, float fov, float aspect);

void mav_gfxOrthogonalSet(float left, float right, float top, float bottom, float nr,
    float fr);
```

```
MAV_matrix *mav_gfxMatrixGet(void);
void mav_gfxPolygonBegin(void);
void mav_gfxPolygonEnd(void);
void mav_gfxTrianglesBegin(void);
void mav_gfxTrianglesEnd(void);
void mav_gfxStripQBegin(void);
void mav_gfxStripQEnd(void);
void mav_gfxStripTBegin(void);
void mav_gfxStripTEnd(void);
void mav_gfxLineClosedBegin(void);
void mav_gfxLineClosedEnd(void);
void mav_gfxLineBegin(void);
void mav_gfxLineEnd(void);
void mav_gfxMeshTBegin(void);
void mav_gfxMeshTEnd(void);
void mav_gfxVertex(MAV_vector v);
void mav_gfxNormal(MAV_vector n);
void mav_gfxTexCoord(MAV_texCoord t);
void mav_gfxColouringModeUse(MAV_palette *p, int mode);
void mav_gfxColourSet(MAV_colour col);
void mav_gfxColourUse(MAV_colour col);
void mav_gfxMaterialSet(MAV_material mat);
void mav_gfxMaterialUse(MAV_material mat);
void mav_gfxTextureSet(MAV_texture *tex, MAV_texEnvFn pTexEnv);
void mav_gfxTextureUse(MAV_texture tex, MAV_texEnvFn pTexEnv);
void mav_gfxLightingModelSet(MAV_lightingModel lm);
void mav_gfxLightingModelUse(MAV_lightingModel lm);
void mav_gfxLightSet(MAV_light l);
void mav_gfxLightUse(MAV_light l);
void mav_gfxLightPos(MAV_light l);
void mav_gfxBlendSet(int v);
void mav_gfxTextureEnv1Set(int v);
void mav_gfxTextureEnv2Set(int v);
void mav_gfxAccumSet(int mode, float val);
int mav_gfxListsNew(int range);
```

void mav_gfxListNew(int list, int mode);

void mav_gfxListEnd(void);

void mav_gfxListExec(int list);

void mav_gfxListsExec(int n, int *lists);

void mav_gfxListsDelete(int list, int range);

void mav_gfx3DfxModeSet(int fullscreen);

int mav_gfx3DfxBoardSet(int bd);

Description

These functions are wrappers to the corresponding graphics system (OpenGL, IrisGL or Direct3D) functions.

---

mav_kernelID

MAVERIK Level 2 functions

Summary

Identifies the kernel.

Syntax

char *mav_kernelID(void);

Description

This function returns an identification string for the MAVERIK kernel, which contains information such as version number. This function could be called directly by the user, for example, to ensure that an application was running with a particular version of MAVERIK .

mav_kernelID is called internally during **mav_initialise**, and the return string printed to `stdout` to form the “welcome message”

---

mav_lineFrom2DPoint

Summary

Compute vector from eye through screen position.

Syntax

MAV_line mav_lineFrom2DPoint(MAV_window *w, int x, int y);

MAV_window *w

Window.

int x

x coordinate (pixel).

int y

y coordinate (pixel).

Description

This function computes and returns a vector from the eyepoint in window `w`, through screen pixel (`x`, `y`).

---

mav_linePolygonIntersection

MAVERIK Level 2 functions

mav_linePolygonIntersection, mav_lineInfPlaneIntersection, mav_lineAxisPlaneIntersection

---

Summary

Geometrical intersection functions.

Syntax

**int mav_linePolygonIntersection(MAV_polygon *p, MAV_line ln,
MAV_objectIntersection *oi);**

**int mav_lineInfPlaneIntersection(MAV_vector pt, MAV_vector norm, MAV_line ln,
MAV_objectIntersection *oi);**

**int mav_lineAxisPlaneIntersection(float xmin, float xmax, float ymin, float ymax,
float zmin, float zmax, MAV_vector pt, MAV_vector norm, MAV_line ln,
MAV_objectIntersection *oi);**

Description

These are utility functions for computing intersections between lines and polygons, and lines and planes. Each function returns MAV_TRUE if an intersection was detected, otherwise MAV_FALSE.

- **mav_linePolygonIntersection**
  tests for intersection between line `ln` and polygon `p`, returning the closest intersection point (if any) in `oi`.

  = item **mav_lineInfPlaneIntersection** tests for intersection between the line `ln` and the plane defined by the point `pt` and the normal vector `norm`.

- **mav_lineAxisPlaneIntersection**
  calculates the intersection of a line `ln` with a finite axis-aligned plane. The plane is defined by a point `pt` and plane normal `norm`. The extent of the plane is defined by providing 3 sets of min/max values, one for each axis. The extent for the two axes over which the plane is defined are set to the appropriate values. The extent for the remaining axis, to which the plane is perpendicular, should be set to +/- `MAV_INFINITY`.

---

mav_lineTransFrame

Summary

Transform world coordinates line to object coordinates.

Syntax

MAV_line mav_lineTransFrame(MAV_line in, MAV_matrix m);

MAV_line in

World coordinates line.

MAV_matrix m

Matrix.

Description

This function takes line `in` in world coordinates, and transforms it using the inverse of matrix `m`, returning the result. The intent of this function is to specify `m` as the matrix of an object, such that the line `in` is transformed from world coordinates into the local coordinates of the object.

See also

mav_drawInfoTransFrame (page 265).

mav_matrixScaleGet

MAVERIK Level 2 functions

Summary

Query scale of matrix.

Syntax

float *mav_matrixScaleGet(MAV_matrix m);

MAV_matrix m

Description

MAVERIK dictates that objects can only be scaled uniformly. This function returns the scale used in matrix `m`, by (arbitrarily) querying the x-scale. This information is useful for applications which wish to write their own object intersection routines.

mav_matrixStackPush

mav_matrixStackPush, mav_matrixStackGet, mav_matrixStackPop

Summary

Maverik matrix stack functions.

Syntax

void mav_matrixStackPush(MAV_matrix m);

MAV_matrix mav_matrixStackGet(void);

void mav_matrixStackPop(void);

MAV_matrix m

Matrix.

Description

MAVERIK maintains a matrix utility stack for use by applications. The stack is private to MAVERIK and is separate from the graphics matrix stack.

mav_matrixStackPush pushes matrix `m` onto the stack.

mav_matrixStackGet returns the matrix on the top of the stack, but does not pop the stack.

mav_matrixStackPop pops the stack.

mav_navigate

MAVERIK Level 2 functions

Summary

Call a navigation function.

Syntax

void mav_navigate(MAV_navigatorFn fn, MAV_viewParams *vp, float am, float ls, float as);

MAV_navigatorFn fn

Navigation function to call.

MAV_viewParams *vp

View parameters to modify.

float am

Amount by which to modify the view parameters.

float ls

Linear scale factor.

float as

Angular scale factor.

Description

This function calls the navigation function `fn`, to modify the view parameters `vp`. It modifies them by amount `am`, using linear scale `ls` to convert `am` into application units, and angular scale `as` to to convert `am` into radians.

mav_navigate is called by the various device-specific navigation functions, in order to perform navigation. Here, `am` will be measured in the intrinsic units of the device (pixels or inches, for example), and the scaling factors `ls` and `as` are provided by the application to map these values into application world-coordinate units.

mav_objectIntersectionsSort

Summary

Find nearest in list of object intersections.

Syntax

int mav_objectIntersectionsSort(int nhits, MAV_objectIntersection *hits, float scale, MAV_objectIntersection *res);

int nhits

Number of intersection hits.

MAV_objectIntersection *hits

Array of intersection hits.

float scale

Scaling factor.

MAV_objectIntersection *res

Closest intersection.

Description

This function takes an array `hits` of `nhits` intersections, and returns in `res` the intersection closest to the origin of the intersecting line. `scale` accounts for non-unity scaling of the intersected object.

mav_objectSMSsGet

MAVERIK Level 2 functions

Summary

Query which SMSs contain a specified object.

Syntax

MAV_list *mav_objectSMSsGet(MAV_object *o);

MAV_object *o

Object to find.

Description

This function efficiently searches through all known MAVERIK SMSs to locate which ones contain the specified object. A list of these SMSs forms the return value of the function. Users should not modify this list in any way. The use of this function requires that object tables are enabled, which it is by default.

See also

mav_objectNew, **mav_objectDataGet**

mav_paletteNew

Summary

Create a new palette.

Syntax

MAV_palette *mav_paletteNew(void);

Description

This function creates a new palette, and returns its handle.

mav_surfaceParamsUndefine

MAVERIK Level 2 functions

Summary

Notify kernel that surface parameters may have changed.

Syntax

void mav_surfaceParamsUndefine(void);

Description

This function informs the kernel that its knowldge of the current surface parameters may be incorrect, because the application may have intervened at the OpenGL level. The kernel responds to this function by not attempting to optimise unnecessary context changes. For advanced use only.

mav_surfaceParamsUse

Summary

Set the surface parameters for the current window.

Syntax

void mav_surfaceParamsUse(MAV_surfaceParams *sp);

MAV_surfaceParams *sp

the surface parameters to use

Description

This function sets the surface parameters (i.e. “what colour”) to use for rendering objects in the current window.

See also

mav_surfaceParamsNew.

---

mav_windowPaletteSet

MAVERIK Level 2 functions

Summary

Set a palette for a window.

Syntax

void mav_windowPaletteSet(MAV_window *w, MAV_palette *p);

MAV_window *w

Window to set.

MAV_palette *p

Palette to set.

Description

This function sets the palette for window `w` to be `p`. Each MAVERIK window has an associated palette, created by default when the window is created. The palette contains a light table, a colour table, a materials table, and a textures table.

Chapter 7

Level 3 functions

---

mav_HBBObjectAdd MAVERIK Level 3 functions

mav_HBBObjectAdd, mav_HBBObjectRmv, mav_HBBIntersect, mav_HBBPointerReset, mav_HBBPointerPush, mav_HBBPointerPop, mav_HBBObjectNext, mav_HBBExecFn, mav_HBBSize, mav_HBBEmpty, mav_HBBDelete, mav_HBBConstructFromSMS

Summary

SMS management: Hierachical Bounding Volume.

Syntax

int mav_HBBObjectAdd(MAV_SMS *s, MAV_object *o);

int mav_HBBObjectRmv(MAV_SMS *s, MAV_object *o);

int mav_HBBIntersect(MAV_SMS *s, MAV_window *w, MAV_line *ln, MAV_objectIntersection *oi, MAV_object **o);

int mav_HBBPointerReset(MAV_SMS *s);

int mav_HBBPointerPush(MAV_SMS *s);

int mav_HBBPointerPop(MAV_SMS *s);

int mav_HBBObjectNext(MAV_SMS *s, MAV_object **o);

int mav_HBBExecFn(MAV_SMS *s, MAV_drawInfo *di, MAV_SMSExecFn *fn);

int mav_HBBSize(MAV_SMS *s, int *sz);

int mav_HBBEmpty(MAV_SMS *s, int *o);

int mav_HBBDelete(MAV_SMS *s, int *o);

void mav_HBBConstructFromSMS(MAV_SMS *target, MAV_SMS *from);

Description

These are the callback routines for the HBB class of SMS. Each of these functions is registered for an SMS of this class on MAVERIK initialisation.

- **mav_HBBObjectAdd**
  adds object `o` to SMS `s`.
- **mav_HBBObjectRmv**
  removes object `o` from SMS `s`.
- **mav_HBBIntersect**
  returns in `o` the object in the SMS with the closest intersection with line `ln`. If an intersection is detected, the closest intersection point is returned in `oi`, and the result of the function is MAV_TRUE. If no intersection is detected, the result of the function is MAV_FALSE.
- **mav_HBBPointerReset**
  resets the SMS pointer to the start of the SMS.
- **mav_HBBObjectNext**
  returns in `o` the object at the current pointer position in the SMS.
- **mav_HBBExecFn**
  executes the callback function `fn`, passing it drawing information `di`.
- **mav_HBBSize**
  returns in `sz` the number of objects in SMS `s`.

mav_SMSCallbackDeleteExec

Summary

Execute SMS delete callback.

Syntax

int mav_SMSCallbackDeleteExec(MAV_SMS *s, int o);

MAV_SMS *s

SMS.

int o

Whether or not to delete objects in SMS.

Description

This function calls the SMS "delete" callback function registered by **mav_SMSCallbackDeleteSet**. `o` controls whether the objects contained in the SMS should also be deleted (`o` = MAV_TRUE for deletion, `o` = MAV_FALSE for no deletion).

See also

mav_SMSCallbackDeleteSet (page 286).

mav_SMSCallbackDeleteSet

MAVERIK Level 3 functions

Summary

Set SMS delete callback.

Syntax

void mav_SMSCallbackDeleteSet(MAV_SMSClass *sc, MAV_SMSCallbackDeleteFn fn);

MAV_SMSClass *sc

SMS class.

MAV_SMSCallbackDeleteFn fn

SMS ”delete” callback function.

Description

This function sets the “delete” callback function for SMS class `c` to be `fn`. The callback is called when the SMS is deleted.

See also

mav_SMSCallbackDeleteExec (page 285).

mav_SMSCallbackEmptySet

mav_SMSCallbackEmptySet, mav_SMSCallbackEmptyExec

Summary

Set/execute SMS callback to remove all objects from SMS.

Syntax

void mav_SMSCallbackEmptySet(MAV_SMSClass *sc, MAV_SMSCallbackEmptyFn fn);

int mav_SMSCallbackEmptyExec(MAV_SMS *s, int o);

Description

mav_SMSCallbackEmptySet sets the “empty SMS” callback function for SMS class `sc` to be `fn`. `fn` is a function which when executed (by **mav_SMSCallbackEmptyExec**), removes all the objects from SMS `s`.

---

mav_SMSCallbackExecFnSet

MAVERIK Level 3 functions

mav_SMSCallbackExecFnSet, mav_SMSCallbackExecFnExec

Summary

Set/execute SMS callback function.

Syntax

void mav_SMSCallbackExecFnSet(MAV_SMSClass *sc, MAV_SMSCallbackExecFnFn fn);

int mav_SMSCallbackExecFnExec(MAV_SMS *s, MAV_drawInfo *di, MAV_SMSExecFn *fn);

Description

mav_SMSCallbackExecFnSet sets a callback function which upon execution uses drawing information `di` to determine for each object in the SMS whether it should be processed further by the function defined in `fn`. Normally this will be a drawing function and the `di` would include the clip planes associated with a given view.

An example of a non-drawing use of `fn` might be to count the number of objects occupying an arbitrary volume of space. Here, `di` would be set to the clip planes defining the volume (and the view information would not be used).

---

mav_SMSCallbackExec

Summary

Execute an SMS callback function.

Syntax

int mav_SMSCallbackExec(MAV_SMSCallback *scb, MAV_SMS *s, void *d1, void *d2, void *d3, void *d4);

MAV_SMSCallback *scb

The callback function to be executed.

MAV_SMS *s

The SMS

void *d1, void *d2, void *d3, void *d4

Data for callback.

Description

The result of this function is the result of the callback function.

---

mav_SMSCallbackIntersectSet

MAVERIK Level 3 functions

mav_SMSCallbackIntersectSet, mav_SMSCallbackIntersectExec

Summary

Set/execute ”Intersect” callback function.

Syntax

void mav_SMSCallbackIntersectSet(MAV_SMSClass *sc, MAV_SMSCallbackIntersectFn fn);

int mav_SMSCallbackIntersectExec(MAV_SMS *s, MAV_window *w, MAV_line ln, MAV_objectIntersection *oi, MAV_object **o);

MAV_SMSClass *sc

Description

- **mav_SMSCallbackIntersectSet**
  sets the callback function for SMS class `sc`, which performs object intersection testing.
- **mav_SMSCallbackIntersectExec**
  executes the SMS callback function registered by **mav_SMSCallbackIntersectSet**. The callback steps through the SMS executing each object's intersection callback when necessary. The closest intersection of line `ln` with an object (if any) is returned in `oi`, and the relevant object in `o`. The result of the function is MAV_TRUE if an intersection was detected, else MAV_FALSE.

---

mav_SMSCallbackNew

Summary

Create new SMS callback function.

Syntax

MAV_SMSCallback *mav_SMSCallbackNew(void);

Description

This function creates a new SMS callback functionand returns its handle. The callback may subsequently be set to be a specific function using **mav_SMSCallbackSet**.

See also

mav_SMSCallbackSet.

---

mav_SMSCallbackObjectAddSet

MAVERIK Level 3 functions

mav_SMSCallbackObjectAddSet, mav_SMSCallbackObjectAddExec

Summary

Set/execute ”add object to SMS” callback function.

Syntax

void mav_SMSCallbackObjectAddSet(MAV_SMSClass *sc, MAV_SMSCallbackObjectAddFn fn);

int mav_SMSCallbackObjectAddExec(MAV_SMS *s, MAV_object *o);

Description

mav_SMSCallbackObjectAddSet sets the “add object to SMS” callback function which will add an object into an SMS of class *sc.

When an application calls **mav_SMSCallbackObjectAddExec**, the “add object to SMS” callback function will add object `o` to SMS `s`.

---

mav_SMSCallbackObjectContainsSet

mav_SMSCallbackObjectContainsSet, mav_SMSCallbackObjectContainsExec

Summary

Set/execute SMS callback to query if SMS contains a specific object.

Syntax

void mav_SMSCallbackObjectContainsSet(MAV_SMSClass *sc,
MAV_SMSCallbackObjectContainsFn fn);

int mav_SMSCallbackObjectContainsExec(MAV_SMS *s, MAV_object *o, int *cnt);

Description

mav_SMSCallbackObjectContainsSet sets the "SMS contains object" callback function for SMS class `sc` to be `fn`. `fn` is a function which when executed (by **mav_SMSCallbackObjectContainsExec**), returns MAV_TRUE in `cnt` if object *o is present in SMS `s`, otherwise MAV_FALSE.

mav_SMSCallbackObjectRmvSet

MAVERIK Level 3 functions

mav_SMSCallbackObjectRmvSet, mav_SMSCallbackObjectRmvExec

Summary

Set/exec SMS ”remove object” callback.

Syntax

void mav_SMSCallbackObjectRmvSet(MAV_SMSClass *sc, MAV_SMSCallbackObjectRmvFn fn);

int mav_SMSCallbackObjectRmvExec(MAV_SMS *s, MAV_object *o);

MAV_SMSClass *sc

Description

- **mav_SMSCallbackObjectRmvSet**
  sets the “remove object” callback function for SMS class `sc` to be `fn`. The callback is called when an object is removed from the SMS.
- **mav_SMSCallbackObjectRmvExec**
  calls the callback function registered by **mav_SMSCallbackObjectRmvSet** for object `o`.

---

mav_SMSCallbackPointerResetSet

mav_SMSCallbackPointerResetSet, mav_SMSCallbackPointerResetExec, mav_SMSCallbackPointerPushSet, mav_SMSCallbackPointerPushExec, mav_SMSCallbackPointerPopSet, mav_SMSCallbackPointerPopExec, mav_SMSCallbackObjectNextSet, mav_SMSCallbackObjectNextExec

---

Summary

Set SMS pointer callback functions.

Syntax

void mav_SMSCallbackPointerResetSet(MAV_SMSClass *sc, MAV_SMSCallbackPointerResetFn fn);

int mav_SMSCallbackPointerResetExec(MAV_SMS *s);

void mav_SMSCallbackPointerPushSet(MAV_SMSClass *sc, MAV_SMSCallbackPointerPushFn fn);

int mav_SMSCallbackPointerPushExec(MAV_SMS *s);

void mav_SMSCallbackPointerPopSet(MAV_SMSClass *sc, MAV_SMSCallbackPointerPopFn fn);

int mav_SMSCallbackPointerPopExec(MAV_SMS *s);

void mav_SMSCallbackObjectNextSet(MAV_SMSClass *sc, MAV_SMSCallbackObjectNextFn fn);

int mav_SMSCallbackObjectNextExec(MAV_SMS *s, MAV_object **o);

MAV_SMSClass *sc

Description

These functions provide a convenient way to step through an SMS. Each SMS has a conceptual "pointer", and a private stack on which to save it.

- **mav_SMSCallbackPointerResetSet**
  sets the "reset pointer" callback function for SMS class `sc`, to be `fn`. `fn` will set the pointer to the beginning of the SMS. **mav_SMSCallbackPointerResetExec** executes this callback for SMS `s`.

- **mav_SMSCallbackPointerPushSet**
  sets the “push pointer” callback function for SMS class `sc`, to be `fn`. `fn` will push the pointer onto the pointer stack. **mav_SMSCallbackPointerPushExec** executes this callback for SMS `s`.

- **mav_SMSCallbackPointerPopSet** and **mav_SMSCallbackPointerPopExec**
  work similarly, but pop the pointer from the pointer stack.

- **mav_SMSCallbackObjectNextSet**
  sets the “get next object” callback function for SMS class `sc`, to be `fn`. `fn` will return in `oo` the object at the current pointer position in SMS `s`. The pointer is then moved onto the next object.

mav_SMSCallbackQuery

Summary

Query an SMS callback function.

Syntax

MAV_SMSCallbackFn mav_SMSCallbackQuery(MAV_SMSCallback *scb, MAV_SMS *s);

MAV_SMSCallback *scb

Callback function to query.

MAV_SMS *s

SMS to query.

Description

This function queries whether SMS `s` currently has a callback function `scb`. The result of the function is the callback function if one is set, otherwise `NULL`.

---

mav_SMSCallbackSet

MAVERIK Level 3 functions

Summary

Set an SMS callback.

Syntax

void mav_SMSCallbackSet(MAV_SMSCallback *scb, MAV_SMSClass *sc, MAV_SMSCallbackFn fn);

MAV_SMSCallback *scb

Handle to callback, as created by **mav_SMSCallbackNew**.

MAV_SMSClass *sc

Class of SMS with which to associate the callback.

MAV_SMSCallbackFn fn

The SMS callback function itself.

Description

This function sets the SMS callback function for SMS class `sc` to be `fn`.

---

mav_SMSCallbackSizeSet

mav_SMSCallbackSizeSet, mav_SMSCallbackSizeExec

Summary

Set/execute SMS callback to return number of objects in SMS.

Syntax

void mav_SMSCallbackSizeSet(MAV_SMSClass *sc, MAV_SMSCallbackSizeFn fn);

int mav_SMSCallbackSizeExec(MAV_SMS *s, int *sz);

Description

mav_SMSCallbackSizeSet sets the “query SMS size” callback function for SMS class `sc` to be `fn`. `fn` is a function which when executed (by **mav_SMSCallbackSizeExec**), returns in `sz` the number of objects in SMS `s`.

mav_SMSClassGet

MAVERIK Level 3 functions

Summary

Query the class of an SMS.

Syntax

MAV_SMSClass *mav_SMSClassGet(MAV_SMS *s);

MAV_SMS *s

SMS to query.

Description

This function returns the class of SMS `s`.

mav_SMSClassNew

Summary

Create new SMS class.

Syntax

MAV_SMSClass *mav_SMSClassNew(void);

Description

This function creates a new SMS class. This is a low-level function, which is called internally by higher level MAVERIK functions in order to create the default SMS's. It would normally only be called directly by applications defining new methods of storing objects.

mav_SMSDataGet

MAVERIK Level 3 functions

Summary

Queries the data associated with an SMS.

Syntax

void *mav_SMSDataGet(MAV_SMS *s);

MAV_SMS *s

SMS to be queried.

Description

This function returns the data associated with SMS `s`.

mav_SMSNew

Summary

Create a new SMS.

Syntax

MAV_SMS *mav_SMSNew(MAV_SMSClass *sc, void *d);

MAV_SMSClass *sc

SMS class.

void *d

Data structure associated with the SMS.

Description

This function creates a new SMS of class `sc`, with associated data structure `d`. `d` is normally the return value of **mav_objListNew** or **mav_HBBNew**

mav_callbackExec

Summary

Execute an object callback function.

Syntax

int mav_callbackExec(MAV_callback *cb, MAV_window *w, MAV_object *o, void *d1, void *d2);

MAV_callback *cb

The callback function to be executed.

MAV_window *w

Window.

MAV_object *o

Object.

void *d1

Callback data 1.

void *d2

Callback data 2.

Description

This function executes the function set for callback `cb` in window `w` on object `o`, passing to it `d1` and `d2`. `d1` and `2` are callback specific data structures cast to the be void pointers.

The return value of this function is the return value of the callback function or MAV_FALSE if no callback function has been set. Interpretion of this value is specific to each callback, but usually indicates the sucess or failure of the operation.

Higher level wrapper function are provided, such as **mav_callbackDrawExec**, which allows for easier use and tigher prototyping.

mav_callbackNew

Summary

Create a new object callback.

Syntax

MAV_callback *mav_callbackNew(void);

Description

This function creates a new object callback and returns its handle. The callback may subsequently be set to be a specific function using **mav_callbackSet**.

See also

mav_callbackSet.

mav_callbackQuery

MAVERIK Level 3 functions

Summary

Query an object callback function.

Syntax

MAV_callbackFn mav_callbackQuery(MAV_callback *cb, MAV_window *w, MAV_object *o);

MAV_callback *cb

Callback function to query.

MAV_window *w

Window to query.

MAV_object *o

Object to query.

Description

This function queries whether object `o` currently has a callback function `cb` set in window `w`. The result of the function is the callback function if one is set, otherwise `NULL`.

mav_callbackSet

Summary

Set a callback.

Syntax

**void mav_callbackSet(MAV_callback *cb, MAV_window *w, MAV_class *c,
MAV_callbackFn fn);**

MAV_callback *cb

Handle to object callback, as created by **mav_callbackNew**.

MAV_window *w

The window with which to associate the callback.

MAV_class *c

Class of object with which to associate the callback.

MAV_callbackFn fn

The callback function itself.

Description

This function sets the callback function for object class `c` in window `w` to be `fn`.

---

mav_callbackSysKeyboardSet

MAVERIK Level 3 functions

mav_callbackSysKeyboardSet, mav_callbackSysKeyboardExec

Summary

Keyboard callback management (system use only).

Syntax

void mav_callbackSysKeyboardSet(MAV_window *w, MAV_class *c,
MAV_callbackKeyboardFn fn);

int mav_callbackSysKeyboardExec(MAV_window *w, MAV_object *o,MAV_keyboardEvent *ke);

Description

These functions operate analagously to **mav_callbackKeyboardSet** and **mav_callbackKeyboardExec**, except that they trap various keypresses to control navigation.

---

mav_callbackSysMouseSet

mav_callbackSysMouseSet, mav_callbackSysMouseExec

Summary

Mouse callback management (system use only).

Syntax

void mav_callbackSysMouseSet(MAV_window *w, MAV_class *c, MAV_callbackMouseFn fn);

int mav_callbackSysMouseExec(MAV_window *w, MAV_object *o,MAV_mouseEvent *me);

Description

These functions operate analagously to **mav_callbackMouseSet** and **mav_callbackMouseExec**, except that they trap various mouse button presses on any button to control navigation.

---

mav_callbacksModuleID

MAVERIK Level 3 functions

mav_callbacksModuleID, mav_gfxModuleID, mav_navigationModuleID, mav_objectsModuleID, mav_SMSModuleID, mav_windowsModuleID, mav_avatarModuleID, mav_TDMModuleID, mav_TRModuleID

Summary

Query kernel module ID.

Syntax

char *mav_callbacksModuleID(void);

char *mav_gfxModuleID(void);

char *mav_navigationModuleID(void);

char *mav_objectsModuleID(void);

char *mav_SMSModuleID(void);

char *mav_windowsModuleID(void);

char *mav_avatarModuleID(void);

char *mav_TDMModuleID(void);

char *mav_TRModuleID(void);

Description

These functions return the identification string for the appropriate kernel module.

---

mav_callbacksModuleInit

mav_callbacksModuleInit, mav_gfxModuleInit, mav_navigationModuleInit, mav_objectsModuleInit, mav_SMSModuleInit, mav_windowsModuleInit, mav_avatarModuleInit, mav_TDMModuleInit, mav_TRModuleInit

Summary

Initialise the supporting modules.

Syntax

int mav_callbacksModuleInit(void);

int mav_gfxModuleInit(void);

int mav_navigationModuleInit(void);

int mav_objectsModuleInit(void);

int mav_SMSModuleInit(void);

int mav_windowsModuleInit(void);

int mav_avatarModuleInit(void);

int mav_TDMModuleInit(void);

int mav_TRModuleInit(void);

Description

These functions initialise the various supporting modules. These functions are normally called internally by **mav_initialise**.

See also

mav_initialise.

mav_deviceNew

MAVERIK Level 3 functions

Summary

Register a new input device.

Syntax

**void mav_deviceNew(MAV_devicePollFn dpfn, MAV_deviceCalcFn dcfn,
MAV_deviceEventFn defn);**

MAV_devicePollFn dpfn

Function to poll device to obtain raw device data.

MAV_deviceCalcFn dcfn

Function to map device coordinates into world coordinates.

MAV_deviceEventFn defn

Function to handle events.

Description

This function registers a new input device.

mav_devicePoll

mav_devicePoll, mav_deviceCalc

Summary

Poll all input devices.

Syntax

void mav_devicePoll(void);

void mav_deviceCalc(void);

Description

- **mav_devicePoll**
  calls the poll function for each registered input device. **mav_devicePoll** is called internally by other MAVERIK routines, such as **mav_frameBegin**.
- **mav_deviceCalc**
  calls the `MAV_deviceCalcFn` function for each registered input device.

mav_gfxWindowBuffersSwap

MAVERIK Level 3 functions

Summary

Swap the buffers.

Syntax

void mav_gfxWindowBuffersSwap(void);

Description

Swaps the buffers for the active graphics context. This routine does nothing for single buffered graphics context.

mav_gfxWindowClose

Summary

Set active rendering window.

Syntax

void mav_gfxWindowClose(int id);

int id

Identifies the window.

Description

Closes window `id`.

---

mav_gfxWindowEventGet

MAVERIK Level 3 functions

Summary

Check for window events.

Syntax

int mav_gfxWindowEventGet(int *info);

int *info

returns information on the event

Description

Thus function checks the window event queue to see if any events are outstanding. The window events that are trapped are key press/release, mouse button press/release, window crossing, resize, mapping and exposure.

If an event is waiting to be processed, it is removed from the queue and details of that event are returned in `info`. The contents of `info` are dependent on the event type and are trivially derived from the source code to **mav_gfxWM**.

The function returns `MAV_TRUE` if an event was outstanding, `MAV_FALSE` otherwise.

---

mav_gfxWindowEventPeek

Summary

Query window events.

Syntax

int mav_gfxWindowEventPeek(void);

Description

This function checks for, but does not remove from the queue, window events.

The routine returns `MAV_TRUE` if an event is outstanding, `MAV_FALSE` otherwise.

---

mav_gfxWindowFontSet

MAVERIK Level 3 functions

Summary

Define a font.

Syntax

void mav_gfxWindowFontSet(char *s, int font, int *width);

char *s

The name of the font to define

int font

The index to refer to the font in future calls

int *width

The width of each character in pixels

Description

mav_gfxWindowKeyGet

Summary

Query the status of a key.

Syntax

int mav_gfxWindowKeyGet(int key);

int key

ASCII value of key in question

Description

Returns `MAV_TRUE` if key is pressed, `MAV_FALSE` otherwise. Hash defines in mav_windows.h are used to identify keys which do not have ASCII values (for example the function keys).

---

mav_gfxWindowOpen

Summary

Open a window.

Syntax

**void mav_gfxWindowOpen(int id, int x, int y, int w, int h, char *name, char *disp,
int wmp, int sb, int qb, int ms, int ab, int stenb, int desta, int *wret,
int *hret);**

int id

A unique handle to identify the window in subsequent calls.

int x, int y

(x,y) position of window.

int w, int h

Width and height of window.

char *name

Name to place on titlebar.

char *disp

Name of X display on which to open window.

int wmp

Requests window manager positioning of window.

int sb

Requests a single buffered graphics context.

int qb

Requests a quad buffered graphics context.

int ms

Requests a multisampled graphics context.

int ab

Requests an accumulation buffered graphics context.

int stenb

Requests a stencil buffered graphics context.

int desta

Requests a destination alpha buffered graphics context.

int *wret, int *hret

Returns the actual width and height of the window opened.

Description

This function asks the window manager on the X display specified by `disp` to open a window and create an associated graphics context. This window is identified in subsequent **mav_gfx*** calls by the unique integer `id`. If `disp` is set to `NULL`, then the display defined by the `DISPLAY` environment variable is used.

Graphics contexts are, by default, double buffered and have the maximum colour and depth bits allowed by the hardware. The parameters `sb`, `qb`, `ms`, `ab`, `stenb` and `desta` (1 = `TRUE`, 0 = `FALSE`) control the creation of the graphics context. If a particular configuration is not supported by the hardware an error message is written to the shell window and execution halts. By default, multiple graphical contexts are "shared" with the first context opened (ie. contexts share a common set of display lists, textures etc).

Windows are of X resource class "MAVERIK App". This allows the potential, depending on your window manager, for controlling window attributes via the .Xdefaults file. For example, adding the line "4Dwm*MAVERIK App*clientDecoration: none" to your .Xdefault would cause MAVERIK windows to have no decoration on SGI's default window manager 4Dwm.

The window manager may not honour your requested size or position. The actual size of the window is returned in `wret` and `hret`. Setting `wmp` to `TRUE` lets the window manager place the window.

mav_gfxWindowPointerGet

MAVERIK Level 3 functions

Summary

Query the position of the mouse.

Syntax

int mav_gfxWindowPointerGet(int id, int *x, int *y, int *rx, int *ry, int *buts);

int id

Identifies the window to measure position relative to.

int *x

Returns horizontal position relative window id.

int *y

Returns vertical position relative window id.

int *rx

Returns horizontal position relative to root window.

int *ry

Returns vertical position relative to root window.

int *buts

Returns the status of the buttons.

Description

This function polls the mouse for its position and button status returning the data in the arguments described above.

mav_gfxWindowPointerSet

Summary

Sets the position of the mouse.

Syntax

void mav_gfxWindowPointerSet(int id, int x, int y);

int id

Identifies the window.

int x

Horizontal position to set.

int y

Vertical position to set.

Description

This function sets the mouse position.

mav_gfxWindowResGet

MAVERIK Level 3 functions

Summary

Query the resolution of the screen.

Syntax

void mav_gfxWindowResGet(int *x, int *y);

int *x

Returns the horizontal resolution.

int *y

Returns the vertical resolution.

Description

This function returns the screen resolution in pixels.

mav_gfxWindowSet

Summary

Set active rendering window.

Syntax

void mav_gfxWindowSet(int id);

int id

Identifies the window.

Description

Sets the graphics context for window `id` to be the active graphics context, which receives and processes the graphics commands. In effect, this function sets the active window for rendering.

mav_gfxWindowStringDisplay

MAVERIK Level 3 functions

Summary

Display a string in a window.

Syntax

void mav_gfxWindowStringDisplay(char *s, int font);

char *s

The string to display

int font

The font to use

Description

mav_moduleDump

Summary

Print details of all kernel modules.

Syntax

void mav_moduleDump(void);

Description

This function calls the identification function for each registered kernel module, printing to `stdout` the result of each function.

mav_moduleNew

MAVERIK Level 3 functions

Summary

Add a new module to the kernel.

Syntax

void mav_moduleNew(MAV_moduleIDFn fn);

MAV_moduleIDFn fn

Module identification function.

Description

This function adds a new module to the kernel. `fn` is an identification function which when called returns an identifier for the module.

mav_objListNew

mav_objListNew, mav_HBBNew

Summary

Create a new SMS.

Syntax

MAV_objList *mav_objListNew(void);

MAV_HBB *mav_HBBNew(void);

Description

These routines return a pointer to a newly created and initialized data structure for each SMS type.

- **mav_objListNew**
  creates a new “object list” SMS, and returns its handle.
- **mav_HBBNew**
  creates a new “hierarchical bounding volume” SMS, and returns its handle.

---

mav_objListObjectAdd

MAVERIK Level 3 functions

mav_objListObjectAdd, mav_objListObjectRmv, mav_objListEmpty, mav_objListIntersect, mav_objListPointerReset, mav_objListPointerPush, mav_objListPointerPop, mav_objListObjectNext, mav_objListExecFn, mav_objListDelete, mav_objListSize

---

Summary

SMS management: object list.

Syntax

int mav_objListObjectAdd(MAV_SMS *s, MAV_object *o);

int mav_objListObjectRmv(MAV_SMS *s, MAV_object *o);

int mav_objListEmpty(MAV_SMS *s, int *o);

int mav_objListIntersect(MAV_SMS *s, MAV_window *w, MAV_line *ln, MAV_objectIntersection *oi, MAV_object **o);

int mav_objListPointerReset(MAV_SMS *s);

int mav_objListPointerPush(MAV_SMS *s);

int mav_objListPointerPop(MAV_SMS *s);

int mav_objListObjectNext(MAV_SMS *s, MAV_object **o);

int mav_objListExecFn(MAV_SMS *s, MAV_drawInfo *di, MAV_SMSExecFn *fn);

int mav_objListDelete(MAV_SMS *s, int *o);

int mav_objListSize(MAV_SMS *s, int *sz);

Description

These are the callback routines for the Object List class of SMS. Each of these functions is registered for an SMS of this class on MAVERIK initialisation.

- **mav_objListObjectAdd**
  adds object `o` to SMS `s`.

- **mav_objListObjectRmv**
  removes object `o` from SMS `s`.

- **mav_objListEmpty**
  empties SMS `s`.

- **mav_objListIntersect**
  returns in `o` the object in the SMS with the closest intersection with line `ln`. If an intersection is detected, the closest intersection point is returned in `oi`, and the result of the function is MAV_TRUE. If no intersection is detected, the result of the function is MAV_FALSE.

- **mav_objListPointerReset**
  resets the SMS pointer to the start of the SMS.

- **mav_objListPointerPush**
  pushes the SMS pointer onto the pointer stack.

- **mav_objListPointerPop**
  pops the SMS pointer off the pointer stack.

- **mav_objListObjectNext**
  returns in `o` the object at the current pointer position in the SMS.

- **mav_objListExecFn**
  executes the callback function `fn`, passing it drawing information `di`.

- **mav_objListDelete**
  deletes object `o` from SMS `s`.

- **mav_objListSize**
  returns in `sz` the number of objects in SMS `s`.

mav_objectTablesSMSAdd

MAVERIK Level 3 functions

Summary

Notify kernel that an object is in an SMS.

Syntax

void mav_objectTablesSMSAdd(MAV_object *o, MAV_SMS *s);

MAV_object *o

Object.

MAV_SMS *s

SMS.

Description

This function notifies the kernel that object `o` has been inserted into SMS `s`. It is intended for use by applications creating and managing their own special kinds of SMS.

See also

mav_objectTablesSMSRmv (page 333).

mav_objectTablesSMSRmv

Summary

Notify kernel that an object is no longer in an SMS.

Syntax

void mav_objectTablesSMSRmv(MAV_object *o, MAV_SMS *s);

MAV_object *o

Object.

MAV_SMS *s

SMS.

Description

This function notifies the kernel that object `o` has been removed from SMS `s`. It is intended for use by applications creating and managing their own special kinds of SMS.

mav_paletteTextureEnvSet

MAVERIK Level 3 functions

Summary

Set a palette texture environment callback.

Syntax

int mav_paletteTextureEnvSet(MAV_palette *p, int index, MAV_texEnvFn fn);

MAV_palette *p

Palette to set.

int index

Index into textures table, in range 0..MAV_MAX_TEXTURES.

MAV_texEnvFn fn

Callback function.

Description

When the kernel detects that the texture environment has been changed (by the application calling **mav_surfaceParamsUse**), it calls a callback function registered by this function. The callback is responsible for setting the environment by calling the appropriate graphics (OpenGL) functions, and is set by default to be **mav_texEnvDefault**.

See also

mav_texEnvDefault (page 339), **mav_surfaceParamsUse** (page 281).

mav_surfaceParamsFlagSet

Summary

Use/ignore surface parameters.

Syntax

void mav_surfaceParamsFlagSet(int use_params);

int use_params

Use or ignore surface parameters.

Description

This function controls whether surface parameters are used to control rendering. If `use_params` = MAV_TRUE, surface parameters are used (the default). If `use_params` = MAV_FALSE, surface parameters are ignored. This function is for advanced use only.

mav_surfaceParamsIsTextured

MAVERIK Level 3 functions

Summary

Query surface parameters for texture.

Syntax

int mav_surfaceParamsIsTextured(MAV_window *w, MAV_surfaceParams *sp);

MAV_window *w

Window with which surface parameters are associated.

MAV_surfaceParams *sp

Surface parameters.

Description

This function examines the surface parameters `sp` on window `w` to determine if textures are specified, returning MAV_TRUE if they are, otherwise MAV_FALSE. This function is used internally by the kernel to minimise context changes during rendering.

mav_surfaceParamsIsTransparent

Summary

Query surface parameters for transparency.

Syntax

int mav_surfaceParamsIsTransparent(MAV_window *w, MAV_surfaceParams *sp);

MAV_window *w

Window with which surface parameters are associated.

MAV_surfaceParams *sp

Surface parameters.

Description

This function examines the alpha components of surface parameters `sp` on window `w`, returning MAV_TRUE if the surface parameters are transparent, otherwise MAV_FALSE. This function is used internally by the kernel to perform rendering of transparent objects at the correct time.

mav_texEnvClamp

MAVERIK Level 3 functions

Summary

A texture environment callback function.

Syntax

void mav_texEnvClamp(MAV_texture *tex);

MAV_texture *tex

Texture to set environment for.

Description

This is a texture environment callback which defines clamped `s` and `t` values and linear minification and magnification filters.

See also

mav_paletteTextureEnvSet.

---

mav_texEnvDefault

Summary

A texture environment callback function.

Syntax

void mav_texEnvDefault(MAV_texture *tex);

MAV_texture *tex

Texture to set environment for.

Description

This is the default texture environment function that handles most common cases. It defines repeating `s` and `t` values and linear minification and magnification filters.

See also

mav_paletteTextureEnvSet.

---

mav_texturedObjectsRender

MAVERIK Level 3 functions

mav_texturedObjectsRender, mav_objectIsTextured, mav_texturedObjectsManage

---

Summary

Efficiently render textured objects.

Syntax

void mav_texturedObjectsRender(void *ignored);

int mav_objectIsTextured(MAV_window *w, MAV_object *o);

void mav_texturedObjectsManage(MAV_window *w, MAV_object *o, MAV_drawInfo *di);

MAV_window *w

Window.

MAV_object *o

Object.

MAV_drawInfo *di

Draw info.

Description

These functions ensure that textured objects are drawn effiently, and are analagous to the functions described in **mav_transparentObjectsRender**.

See also

mav_transparentObjectsRender (page 341).

---

mav_transparentObjectsRender

mav_transparentObjectsRender, mav_objectIsTransparent, mav_transparentObjectsManage

Summary

Render transparent objects correctly.

Syntax

void mav_transparentObjectsRender(void *ignored);

int mav_objectIsTransparent(MAV_window *w, MAV_object *o);

void mav_transparentObjectsManage(MAV_window *w, MAV_object *o, MAV_drawInfo *di);

MAV_window *w

Window.

MAV_object *o

Object.

MAV_drawInfo *di

Draw info.

Description

These functions ensure that transparent objects are drawn correctly.

- **mav_transparentObjectsRender**
  renders all transparent objects.
- **mav_objectIsTransparent**
  returns MAV_TRUE if object `o` in window `w` is transparent, otherwise MAV_FALSE.
- **mav_transparentObjectsManage**
  This function is called when the draw callback is executed for a transparent object `o`. This allows for special treatment, specifically to delay rendering until after all other objects have been drawn, and are depth sorted. These measures are necessary for transparent objects to be displayed correctly.

---

mav_transparentTextRender

MAVERIK Level 3 functions

mav_transparentTextRender, mav_transparentTextManage

Summary

Render transparent text correctly.

Syntax

void mav_transparentTextRender(void *ignored);

void mav_transparentTextManage(MAV_window *w, char *s, int col, int font, float x, float y);

MAV_window *w

Window.

char *s

The text.

int col

The colour.

int font

The font.

float x

The horizontal position.

float y

The vertical position.

Description

These functions ensure that transparent text is drawn correctly.

- **mav_transparentTextRender**
  renders all transparent text.

- **mav_transparentTextManage**
  This function is called when transparent text is rendered. This allows for special treatment, specifically to delay rendering until after all other objects have been drawn. These measures are necessary for transparent text to be displayed correctly.

Part III

MAVERIK constant specifications

Chapter 8

Constants and macros organised by usage

Miscellaneous

MAVERIK Constants and macros

MAV_TRUE	1
MAV_FALSE	0
MAV_DONTCARE	2
MAV_UNDEFINED	-1
MAV_MAX_CBS	100
MAV_MAX_CLIP_PLANES	10
MAV_MAX_WIN	10
MAV_SILENT	0
MAV_VERBOSE	1
MAV_REDEFINE_WARN	1
MAV_REDEFINE_NOWARN	2
MAV_STEREO_TWO_WINS	1
MAV_STEREO_QUAD_BUFFERS	2
MAV_STEREO_QUAD_BUFFERS_SEPARATE_Z	3
MAV_VERSION(X,Y)	((X)*256+(Y))
MAV_THIS_VERSION	the current MAVERIK version as calculated by MAV_VERSION

The version macro can be used at compile time as follows:

```
#if (MAV_THIS_VERSION == MAV_VERSION(5,1))
 <code>
#elif (MAV_THIS_VERSION == MAV_VERSION(5,2))
 <code>
#elif (MAV_THIS_VERSION == MAV_VERSION(5,3))
```

```
 <code>
#endif
```

Mathematics

MAVERIK Constants and macros

MAV_EPSILON	0.001
MAV_INFINITY	1.0E+20
MAV_PI_OVER_180	0.017453292
MAV_180_OVER_PI	57.29578
MAV_PI	3.1415927
MAV_PI_OVER_2	1.5707963
MAV_2_PI	6.2831853
MAV_DEG2RAD(X)	((X)*MAV_PI_OVER_180))
MAV_RAD2DEG(X)	((X)*MAV_180_OVER_PI))

MAV_ID_MATRIX	an identity matrix
MAV_ID_QUATERNION	[1,(0,0,0)]
MAV_NULL_VECTOR	(0,0,0)
MAV_X_VECTOR	(1,0,0)
MAV_Y_VECTOR	(0,1,0)
MAV_Z_VECTOR	(0,0,1)

MAV_MATRIX_XCOMP	0][3
MAV_MATRIX_YCOMP	1][3
MAV_MATRIX_ZCOMP	2][3
MAV_MATRIX_XAXIS_X	0][0
MAV_MATRIX_XAXIS_Y	1][0
MAV_MATRIX_XAXIS_Z	2][0
MAV_MATRIX_YAXIS_X	0][1
MAV_MATRIX_YAXIS_Y	1][1
MAV_MATRIX_YAXIS_Z	2][1
MAV_MATRIX_ZAXIS_X	0][2
MAV_MATRIX_ZAXIS_Y	1][2
MAV_MATRIX_ZAXIS_Z	2][2

Rendering

MAVERIK Constants and macros

MAV_COLOUR	1
MAV_MATERIAL	2
MAV_TEXTURE	3
MAV_LIT_TEXTURE	4
MAV_BLENDED_TEXTURE	5
MAV_COLOUR_BLACK	-10
MAV_COLOUR_WHITE	-11
MAV_COLOUR_RED	-12
MAV_COLOUR_GREEN	-13
MAV_COLOUR_BLUE	-14
MAV_LIGHT_RELATIVE	0
MAV_LIGHT_ABSOLUTE	1

Objects

MAVERIK Constants and macros

MAV_CENTER_JUSTIFY	1
MAV_LEFT_JUSTIFY	2
MAV_RIGHT_JUSTIFY	3
MAV_STROKE_FONT	1
MAV_OUTLINE_FONT	2
MAV_FILLED_FONT	3

MAV_BB_FAST	1
MAV_BB_ACCURATE	2

Events

MAVERIK Constants and macros

MAV_LEFT_BUTTON	0
MAV_MIDDLE_BUTTON	1
MAV_RIGHT_BUTTON	2
MAV_WHEELUP_BUTTON	3
MAV_WHEELDOWN_BUTTON	4
MAV_ANY_BUTTON	20

MAV_PRESSED	0
MAV_RELEASED	1

MAV_MAP	0
MAV_UNMAP	1

MAV_ENTER	0
MAV_LEAVE	1

Non-ASCII key identifiers

MAV_KEY_F1	300
MAV_KEY_F2	301
MAV_KEY_F3	302
MAV_KEY_F4	303
MAV_KEY_F5	304
MAV_KEY_F6	305
MAV_KEY_F7	306
MAV_KEY_F8	307
MAV_KEY_F9	308
MAV_KEY_F10	309
MAV_KEY_F11	310
MAV_KEY_F12	311
MAV_KEY_UP	312
MAV_KEY_DOWN	313
MAV_KEY_LEFT	314
MAV_KEY_RIGHT	315
MAV_KEY_PAGE_UP	316
MAV_KEY_PAGE_DOWN	317
MAV_KEY_SHIFT_L	318
MAV_KEY_SHIFT_R	319
MAV_KEY_ALT_L	320
MAV_KEY_ALT_R	321
MAV_KEY_META_L	322
MAV_KEY_META_R	323
MAV_KEY_HOME	324
MAV_KEY_END	325
MAV_KEY_INSERT	326
MAV_KEY_CTRL_L	327
MAV_KEY_CTRL_R	328
MAV_KEY_CAPS_LOCK	329

Modifier key identifiers

MAVERIK Constants and macros

MAV_MODIFIER_MAX	3
MAV_MODIFIER_SHIFT	0
MAV_MODIFIER_CTRL	1
MAV_MODIFIER_ALT	2

Graphics

MAV_PROJECTION	1
MAV_MODELVIEW	2
MAV_PROJANDVIEW	3
MAV_FRONT	1
MAV_BACK	2
MAV_BLEND_OFF	0
MAV_BLEND_1	1
MAV_POLYGON_LINE	0
MAV_POLYGON_FILL	1
MAV_ACCUM_ACCUM	1
MAV_ACCUM_LOAD	2
MAV_ACCUM_RETURN	3
MAV_ACCUM_ADD	4
MAV_ACCUM_MULT	5
MAV_DLISTS_COMPILE	1
MAV_DLISTS_COMPILE_AND_EXECUTE	2

Functions index

Page numbers in **bold** indicate the main definition page for the function.

calloc, 149

exit, 149

free, 149

malloc, 149
mav_avatarBB, **241**
mav_avatarDraw, **243**
mav_avatarDump, **245**
mav_avatarGetMatrix, **248**
mav_avatarGetSurfaceParams, **250**
mav_avatarGetUserdef, **252**
mav_avatarID, **253**
mav_avatarIntersect, **255**
mav_avatarModuleID, **310**
mav_avatarModuleInit, **311**

mav_BBAlign, **233**, 234
mav_BBCompBB, **233**, 234
mav_BBCompInit, **233**, 234
mav_BBCompPt, **233**, 234
mav_BBCull, **123**, 234
mav_BBCullToClipPlanes, **233**, 234
mav_BBDisplay, **123**
mav_BBDisplayWithColour, **123**
mav_BBDisplayWithSurfaceParams, **123**, 124
mav_BBGetCorner, **233**, 234
mav_BBInsideBB, **233**, 234
mav_BBIntersectsBB, **233**, 234
mav_BBIntersectsClipPlanes, **233**, 234
mav_BBIntersectsLine, **233**, 234
mav_BBPrint, **125**

mav_boxBB, **241**, 242
mav_boxBB2, 242
mav_boxDraw, **243**
mav_boxDump, **245**
mav_boxGetMatrix, **247**
mav_boxGetSurfaceParams, **249**
mav_boxGetUserdef, **251**
mav_boxID, **253**
mav_boxIntersect, **255**

mav_callbackBBExec, **259**
mav_callbackBBSet, **260**
mav_callbackCrossingExec, **261**, 262
mav_callbackCrossingSet, 72, **261**, 262
mav_callbackDeleteExec, **257**, 258
mav_callbackDeleteSet, 178, 257, **258**
mav_callbackDrawExec, **259**, 304
mav_callbackDrawSet, 244, **260**
mav_callbackDumpExec, **259**
mav_callbackDumpSet, **260**
mav_callbackExec, **304**
mav_callbackExposeExec, **261**, 262
mav_callbackExposeSet, 74, **261**, 262
mav_callbackGetMatrixExec, **259**
mav_callbackGetMatrixSet, **260**
mav_callbackGetSurfaceParamsExec, **259**
mav_callbackGetSurfaceParamsSet, **260**
mav_callbackGetUserdefExec, **259**
mav_callbackGetUserdefSet, **260**
mav_callbackIDExec, **259**
mav_callbackIDSet, **260**
mav_callbackIntersectExec, **259**
mav_callbackIntersectSet, **260**
mav_callbackKeyboardExec, **132**, 308
mav_callbackKeyboardSet, 17, **132**, 308
mav_callbackMapExec, **261**, 262
mav_callbackMapSet, 79, **261**
mav_callbackMouseExec, **133**, 309
mav_callbackMouseSet, 20, **133**, 309
mav_callbackNew, **305**, 307
mav_callbackQuery, **306**

mav_callbackResizeExec, **261**
mav_callbackResizeSet, 88, **261**
mav_callbackSet, 305, **307**
mav_callbacksModuleID, **310**
mav_callbacksModuleInit, **311**
mav_callbackSysKeyboardExec, **308**
mav_callbackSysKeyboardSet, **308**
mav_callbackSysMouseExec, **309**
mav_callbackSysMouseSet, **309**
mav_calloc, **149**
MAV_class, 85
mav_classNew, **263**
mav_clipPlanePrint, **125**
mav_clipPlanesGet, **264**
mav_clipPlanesGetFromBB, **264**
mav_clipPlanesGetFromPixels, **264**
mav_clipPlanesPrint, **125**
mav_compositeBB, **241**
mav_compositeDraw, **243**
mav_compositeDump, **245**
mav_compositeEmpty, **134**
mav_compositeGetMatrix, **247**
mav_compositeGetSurfaceParams, **249**
mav_compositeGetUserdef, **251**
mav_compositeID, **253**
mav_compositeIntersect, **255**
mav_compositeRead, 8, **135**
mav_compositeReadAC3D, 134, **135**
mav_compositeReadAC3DBuf, **135**
mav_compositeReadJIF, **135**
mav_compositeReadLWO, **135**
mav_compositeReadVRML97, **135**
mav_coneBB, **241**
mav_coneDraw, **243**
mav_coneDump, **245**
mav_coneGetMatrix, **247**
mav_coneGetSurfaceParams, **249**
mav_coneGetUserdef, **251**
mav_coneID, **253**
mav_coneIntersect, **255**
mav_ctorusBB, **241**
mav_ctorusDraw, **243**
mav_ctorusDump, **245**
mav_ctorusGetMatrix, **247**
mav_ctorusGetSurfaceParams, **249**
mav_ctorusGetUserdef, **251**
mav_ctorusID, **253**
mav_ctorusIntersect, **255**
mav_cylinderBB, **241**
mav_cylinderDraw, **243**
mav_cylinderDump, **245**
mav_cylinderGetMatrix, **247**
mav_cylinderGetSurfaceParams, **249**
mav_cylinderGetUserdef, **251**
mav_cylinderID, **253**
mav_cylinderIntersect, **255**

mav_deviceCalc, **313**
mav_deviceNew, 137, 139, 140, **312**
mav_devicePoll, **313**
mav_drawInfoTransFrame, **265**, 273

mav_ellipseBB, **241**
mav_ellipseDraw, **243**
mav_ellipseDump, **245**
mav_ellipseGetMatrix, **247**
mav_ellipseGetSurfaceParams, **249**
mav_ellipseGetUserdef, **251**
mav_ellipseID, **253**
mav_ellipseIntersect, **255**
mav_eventsCheck, 132, 133, **136**
mav_exposeDefault, **261**, 262
mav_eyeLeft, 35, **138**, 230
mav_eyeMono, **138**, 230
mav_eyeRight, 35, **138**, 230

mav_facetBB, **241**
mav_facetDraw, **243**
mav_facetDump, **245**
mav_facetGetMatrix, **247**
mav_facetGetSurfaceParams, **249**
mav_facetGetUserdef, **251**
mav_facetID, **253**
mav_facetIntersect, **255**
mav_frameBegin, **139**, 141–143, 313
mav_frameEnd, 140, **141**, 142
mav_frameFn0Add, 139, 140, **142**
mav_frameFn0Rmv, **142**
mav_frameFn1Add, 139, 140, **142**
mav_frameFn1Rmv, **142**
mav_frameFn2Add, 139, 140, **142**
mav_frameFn2Rmv, **142**
mav_frameFn3Add, 141, **142**
mav_frameFn3Rmv, **142**
mav_frameFn4Add, 141, **142**

mav_frameFn4Rmv, **142**
mav_frameFnNAdd, 142
mav_frameFnNRmv, 142
mav_free, **149**

mav_gfx, 321
mav_gfx3DfxBoardSet, **269**
mav_gfx3DfxModeSet, **269**
mav_gfxAccumSet, **268**
mav_gfxBackfaceCullGet, **267**
mav_gfxBackfaceCullSet, **267**
mav_gfxBackgroundColourSet, **267**
mav_gfxBlendSet, **268**
mav_gfxBufferReadSet, **267**
mav_gfxClearA, **267**
mav_gfxClearC, **266**
mav_gfxClearCZ, **267**
mav_gfxClearZ, **266**
mav_gfxClipPlaneDisable, **266**
mav_gfxClipPlaneEnable, **266**
mav_gfxClipPlaneSet, **266**
mav_gfxClipPlanesSet, **266**
mav_gfxColouringModeUse, **268**
mav_gfxColourSet, **268**
mav_gfxColourUse, **268**
mav_gfxDepthMaskSet, **267**
mav_gfxDepthTestSet, **267**
mav_gfxFinish, **267**
mav_gfxFlush, **267**
mav_gfxFogSet, **267**
mav_gfxLightingModelSet, **268**
mav_gfxLightingModelUse, **268**
mav_gfxLightPos, **268**
mav_gfxLightSet, **268**
mav_gfxLightUse, **268**
mav_gfxLineBegin, **268**
mav_gfxLineClosedBegin, **268**
mav_gfxLineClosedEnd, **268**
mav_gfxLineEnd, **268**
mav_gfxLineStippleSet, **267**
mav_gfxLineWidthGet, **267**
mav_gfxLineWidthSet, **267**
mav_gfxListEnd, **269**
mav_gfxListExec, **269**
mav_gfxListNew, **269**
mav_gfxListsDelete, **269**
mav_gfxListsExec, **269**
mav_gfxListsNew, **268**
mav_gfxMaterialSet, **268**
mav_gfxMaterialUse, **268**
mav_gfxMatrixGet, **268**
mav_gfxMatrixLoad, **267**
mav_gfxMatrixMode, **267**
mav_gfxMatrixMult, **267**
mav_gfxMatrixPop, **267**
mav_gfxMatrixPush, **267**
mav_gfxMatrixScale, **267**
mav_gfxMatrixTranslate, **267**
mav_gfxMeshTBegin, **268**
mav_gfxMeshTEnd, **268**
mav_gfxModuleID, **310**
mav_gfxModuleInit, **311**
mav_gfxMultiSampleSet, **267**
mav_gfxNormal, **268**
mav_gfxNormalizeSet, **267**
mav_gfxOrthogonalSet, **267**
mav_gfxPerspectiveSet, **267**
mav_gfxPixelDraw, **267**
mav_gfxPixelRead, **267**
mav_gfxPixelReadUByte, **267**
mav_gfxPolygonBegin, **268**
mav_gfxPolygonEnd, **268**
mav_gfxPolygonModeSet, **267**
mav_gfxRasterPos2DSet, **267**
mav_gfxRasterPosSet, **267**
mav_gfxStripQBegin, **268**
mav_gfxStripQEnd, **268**
mav_gfxStripTBegin, **268**
mav_gfxStripTEnd, **268**
mav_gfxTexCoord, **268**
mav_gfxTextureEnv1Set, **268**
mav_gfxTextureEnv2Set, **268**
mav_gfxTextureSet, **268**
mav_gfxTextureUse, **268**
mav_gfxTrianglesBegin, **268**
mav_gfxTrianglesEnd, **268**
mav_gfxVertex, **268**
mav_gfxViewPortSet, **267**
mav_gfxVisualInfoGet, **267**
mav_gfxWindowBuffersSwap, **314**
mav_gfxWindowClose, **315**
mav_gfxWindowEventGet, **316**
mav_gfxWindowEventPeek, **317**
mav_gfxWindowFontSet, **318**

mav_gfxWindowKeyGet, **319**
mav_gfxWindowOpen, **320**
mav_gfxWindowPointerGet, **322**
mav_gfxWindowPointerSet, **323**
mav_gfxWindowResGet, **324**
mav_gfxWindowSet, **325**
mav_gfxWindowStringDisplay, **326**
mav_gfxWM, 316

mav_HBBConstructFromSMS, **284**
mav_HBBDelete, **283**
mav_HBBEmpty, **283**
mav_HBBExecFn, **283**, 284
mav_HBBIntersect, **283**, 284
mav_HBBNew, 39, 303, **329**
mav_HBBObjectAdd, **283**, 284
mav_HBBObjectNext, **283**, 284
mav_HBBObjectRmv, **283**, 284
mav_HBBPointerPop, **283**
mav_HBBPointerPush, **283**
mav_HBBPointerReset, **283**, 284
mav_HBBSize, **283**, 284

mav_hellipseBB, **241**
mav_hellipseDraw, **243**
mav_hellipseDump, **245**
mav_hellipseGetMatrix, **247**
mav_hellipseGetSurfaceParams, **249**
mav_hellipseGetUserdef, **251**
mav_hellipseID, **253**
mav_hellipseIntersect, **255**
mav_hsphereBB, **241**
mav_hsphereDraw, **243**
mav_hsphereDump, **245**
mav_hsphereGetMatrix, **247**
mav_hsphereGetSurfaceParams, **249**
mav_hsphereGetUserdef, **251**
mav_hsphereID, **253**
mav_hsphereIntersect, **255**

mav_initialise, **144**, 183, 242, 270, 311
mav_initialiseNoArgs, **144**

mav_kernelID, **270**
mav_keyboardGet, **145**

mav_lineAxisPlaneIntersection, **272**
mav_lineFrom2DPoint, **271**
mav_lineInfPlaneIntersection, **272**
mav_linePolygonIntersection, **272**
mav_linePrint, **125**
mav_lineTransFrame, 265, **273**
mav_listDelete, **147**, 148
mav_listEmpty, **147**, 148
mav_listItemAdd, 146, **147**
mav_listItemContains, 146, **147**, 148
mav_listItemNext, 146, **147**, 148
mav_listItemRmv, 146, **147**
mav_listItemsAdd, **146**
mav_listItemsContains, **146**
mav_listItemsNext, **146**
mav_listItemsRmv, **146**
mav_listNew, 146, **147**
mav_listPointerPop, **147**, 148
mav_listPointerPush, **147**, 148
mav_listPointerReset, **147**, 148
mav_listPrint, **125**
mav_listSize, **147**, 148

mav_malloc, **149**
mav_mapDefault, **261**, 262
mav_matrixInverse, **150**
mav_matrixMult, **151**
mav_matrixPrint, **125**
mav_matrixQuaternionConvert, **152**
mav_matrixRPYGet, **153**
mav_matrixRPYSet, **154**
mav_matrixScaleGet, **274**
mav_matrixScaleSet, **155**
mav_matrixSet, 81, **156**
mav_matrixStackGet, **275**
mav_matrixStackPop, **275**
mav_matrixStackPush, **275**
mav_matrixXAxisGet, **157**
mav_matrixXAxisSet, **158**
mav_matrixXYZGet, **159**
mav_matrixXYZSet, **160**
mav_matrixYAxisGet, **157**
mav_matrixYAxisSet, **158**
mav_matrixZAxisGet, **157**
mav_matrixZAxisSet, **158**
mav_moduleDump, **327**
mav_moduleNew, **328**
mav_mouseDraw, **161**, 163
mav_mouseGet, **162**

mav_mouseSurfaceParamsSet, **163**

mav_navigate, **276**
mav_navigateForwards, **164**, 165
mav_navigateForwardsFixedUp, **164**, 165
mav_navigateNull, **164**, 165
mav_navigatePitch, **164**, 165
mav_navigatePitchFixedUp, **164**, 165
mav_navigateRight, **164**, 165
mav_navigateRightFixedUp, **164**, 165
mav_navigateRoll, **164**, 165
mav_navigateRotRight, **164**, 165
mav_navigateRotUp, **164**, 165
mav_navigateTransX, **164**, 165
mav_navigateTransY, **164**, 165
mav_navigateTransZ, **164**, 165
mav_navigateUp, **164**, 165
mav_navigateUpFixedUp, **164**, 165
mav_navigateYaw, **164**, 165, 171
mav_navigateYawFixedUp, **164**, 166
mav_navigationKeyboard, 168, 169, **170**, 174
mav_navigationKeyboardDefault, **168**, 170, 173
mav_navigationKeyboardDefaultParams, **167**, 168, 169
mav_navigationModuleID, **310**
mav_navigationModuleInit, **311**
mav_navigationMouse, 170, 173, **174**
mav_navigationMouseDefault, 169, **173**, 174
mav_navigationMouseDefaultParams, **171**, 173
MAV_navigatorFn, 171, 172

MAV_object, 111, 179
mav_objectClassGet, 85, **175**, 176, 179
mav_objectDataGet, 85, **176**, 177, 179, 278
mav_objectDataWith, 176, **177**, 179
mav_objectDelete, **178**, 179
mav_objectIntersectionPrint, **125**
mav_objectIntersectionsSort, **277**
mav_objectIsTextured, **340**
mav_objectIsTransparent, **341**
mav_objectNew, 85, 175–177, **179**, 278
mav_objectsModuleID, **310**
mav_objectsModuleInit, **311**
mav_objectSMSsGet, **278**
mav_objectTablesSMSAdd, **332**
mav_objectTablesSMSRmv, 332, **333**
mav_objListDelete, **330**, 331
mav_objListEmpty, **330**, 331
mav_objListExecFn, **330**, 331
mav_objListIntersect, **330**, 331
mav_objListNew, 83, 303, **329**
mav_objListObjectAdd, **330**
mav_objListObjectNext, **330**, 331
mav_objListObjectRmv, **330**
mav_objListPointerPop, **330**, 331
mav_objListPointerPush, **330**, 331
mav_objListPointerReset, **330**, 331
mav_objListSize, **330**, 331

mav_paletteColourIndexEmptyGet, **180**
mav_paletteColourIndexMatchGet, **181**
mav_paletteColourIndexWarnSet, **198**
mav_paletteColourSet, **183**, 190, 197, 208
mav_paletteColourWarnSet, **198**
mav_paletteFontIndexEmptyGet, **180**
mav_paletteFontIndexMatchGet, **181**
mav_paletteFontIndexWarnSet, **198**
mav_paletteFontSet, **184**
mav_paletteFontWarnSet, **198**
mav_paletteLightIndexEmptyGet, **180**
mav_paletteLightIndexMatchGet, **181**
mav_paletteLightIndexWarnSet, **198**
mav_paletteLightingModelSet, **188**
mav_paletteLightingModelWarnSet, **198**
mav_paletteLightPos, **185**
mav_paletteLightPositioning, **186**
mav_paletteLightSet, **187**
mav_paletteLightWarnSet, **198**
mav_paletteMaterialIndexEmptyGet, **180**
mav_paletteMaterialIndexMatchGet, **181**
mav_paletteMaterialIndexWarnSet, **198**
mav_paletteMaterialSet, 183, **189**, 197, 208
mav_paletteMaterialWarnSet, **198**
mav_paletteNew, **279**
mav_paletteTextureAlphaSet, **191**, 192
mav_paletteTextureColourAlphaSet, **192**
mav_paletteTextureEnvPaletteSet, **193**
mav_paletteTextureEnvSet, 193, **334**, 338, 339
mav_paletteTextureFree, **194**, 196
mav_paletteTextureIndexEmptyGet, **180**
mav_paletteTextureIndexMatchGet, **181**
mav_paletteTextureIndexWarnSet, **198**
mav_paletteTextureMipmappingSet, **195**
mav_paletteTextureSet, 183, 190, **197**, 208

mav_paletteTextureSetFromMem, 194, **196**
mav_paletteTextureWarnSet, **198**
mav_paletteWarnSet, **198**
mav_polygonBB, **241**
mav_polygonDraw, **243**
mav_polygonDump, **245**
mav_polygonGetMatrix, **247**
mav_polygonGetSurfaceParams, **249**
mav_polygonGetUserdef, **251**
mav_polygonGrpBB, **241**
mav_polygonGrpDraw, **243**
mav_polygonGrpDump, **245**
mav_polygonGrpGetMatrix, **247**
mav_polygonGrpGetSurfaceParams, **249**
mav_polygonGrpGetUserdef, **251**
mav_polygonGrpID, **253**
mav_polygonGrpIntersect, **255**
mav_polygonID, **253**
mav_polygonIntersect, **255**
mav_polylineBB, **241**
mav_polylineDraw, **243**
mav_polylineDump, **245**
mav_polylineGetMatrix, **247**
mav_polylineGetSurfaceParams, **249**
mav_polylineGetUserdef, **251**
mav_polylineID, **253**
mav_pyramidBB, **241**
mav_pyramidDraw, **243**
mav_pyramidDump, **245**
mav_pyramidGetMatrix, **247**
mav_pyramidGetSurfaceParams, **249**
mav_pyramidGetUserdef, **251**
mav_pyramidID, **253**
mav_pyramidIntersect, **255**

mav_quaternionInterpolate, **200**
mav_quaternionMatrixConvert, **201**
mav_quaternionPrint, **125**
mav_quaternionSet, 87, **202**

mav_random, **203**
mav_randomSeed, **203**
mav_rectangleBB, **241**
mav_rectangleDraw, **243**
mav_rectangleDump, **245**
mav_rectangleGetMatrix, **247**
mav_rectangleGetSurfaceParams, **249**
mav_rectangleGetUserdef, **251**
mav_rectangleID, **253**
mav_rectangleIntersect, **255**
mav_resizeDefault, **261**
mav_rtorusBB, **241**
mav_rtorusDraw, **243**
mav_rtorusDump, **245**
mav_rtorusGetMatrix, **247**
mav_rtorusGetSurfaceParams, **249**
mav_rtorusGetUserdef, **251**
mav_rtorusID, **253**
mav_rtorusIntersect, **255**

mav_sleep, **204**

mav_SMS_displayFn, 235
mav_SMSCallbackDeleteExec, **285**, 286
mav_SMSCallbackDeleteSet, 285, **286**
mav_SMSCallbackEmptyExec, **287**
mav_SMSCallbackEmptySet, **287**
mav_SMSCallbackExec, **289**
mav_SMSCallbackExecFnExec, **288**
mav_SMSCallbackExecFnSet, **288**
mav_SMSCallbackIntersectExec, **290**
mav_SMSCallbackIntersectSet, **290**
mav_SMSCallbackNew, **291**, 298
mav_SMSCallbackObjectAddExec, 129, **292**
mav_SMSCallbackObjectAddSet, **292**
mav_SMSCallbackObjectContainsExec, **293**
mav_SMSCallbackObjectContainsSet, **293**
mav_SMSCallbackObjectNextExec, **295**
mav_SMSCallbackObjectNextSet, **295**, 296
mav_SMSCallbackObjectRmvExec, 130, **294**
mav_SMSCallbackObjectRmvSet, **294**
mav_SMSCallbackPointerPopExec, **295**, 296
mav_SMSCallbackPointerPopSet, **295**, 296
mav_SMSCallbackPointerPushExec, **295**, 296
mav_SMSCallbackPointerPushSet, **295**, 296
mav_SMSCallbackPointerResetExec, **295**
mav_SMSCallbackPointerResetSet, **295**
mav_SMSCallbackQuery, **297**
mav_SMSCallbackSet, 291, **298**
mav_SMSCallbackSizeExec, **299**
mav_SMSCallbackSizeSet, **299**
MAV_SMSClass, 111
mav_SMSClassGet, 111, **300**
mav_SMSClassNew, **301**

mav_SMSDataGet, 111, **302**
mav_SMSDelete, **126**
mav_SMSDisplay, 73, **127**, 235
mav_SMSDisplayFn, **235**
mav_SMSDisplayUnCulled, **127**
mav_SMSDisplayUsingDrawInfo, **236**
mav_SMSHBBNew, **128**
mav_SMSIntersectBB, **238**
mav_SMSIntersectBBAll, 131, **237**
mav_SMSIntersectLine, **240**
mav_SMSIntersectLineAll, 131, **239**
mav_SMSModuleID, **310**
mav_SMSModuleInit, **311**
mav_SMSNew, 39, 83, 111, **303**
mav_SMSObjBB, **241**
mav_SMSObjDraw, **243**
mav_SMSObjDump, **245**
mav_SMSObjectAdd, **129**, 130
mav_SMSObjectRmv, 129, **130**
mav_SMSObjGetMatrix, **247**
mav_SMSObjGetSurfaceParams, **250**
mav_SMSObjGetUserdef, **251**
mav_SMSObjID, **253**
mav_SMSObjIntersect, **255**
mav_SMSObjListNew, **128**
mav_SMSSelectabilitySet, 111, **131**

mav_sphereBB, **241**
mav_sphereDraw, **243**
mav_sphereDump, **245**
mav_sphereGetMatrix, **247**
mav_sphereGetSurfaceParams, **249**
mav_sphereGetUserdef, **251**
mav_sphereID, **253**
mav_sphereIntersect, **255**
mav_stringDisplay, **205**
mav_stringDisplayCentre, **205**
mav_stringDisplayLeft, **205**
mav_stringDisplayRight, **205**
mav_stringLength, **206**
MAV_surfaceParams, 208
mav_surfaceParamsFlagSet, **335**
mav_surfaceParamsIsTextured, **336**
mav_surfaceParamsIsTransparent, **337**
mav_surfaceParamsNew, 30, **207**, 281
mav_surfaceParamsPrint, **125**
mav_surfaceParamsUndefine, **280**
mav_surfaceParamsUse, 207, 208, **281**, 334

mav_TDMCursorBB, **241**
mav_TDMCursorDraw, **244**
mav_TDMCursorDump, **246**
mav_TDMCursorGetSurfaceParams, **250**
mav_TDMCursorGetUserdef, **252**
mav_TDMCursorID, **253**
mav_TDMModuleID, **310**
mav_TDMModuleInit, **311**

mav_teapotBB, **241**
mav_teapotDraw, **243**
mav_teapotDump, **245**
mav_teapotGetMatrix, **247**
mav_teapotGetSurfaceParams, **250**
mav_teapotGetUserdef, **251**
mav_teapotID, **253**
mav_texCoordPrint, **125**
mav_texEnvClamp, **338**
mav_texEnvDefault, 334, **339**
mav_textBB, **241**
mav_textDraw, **243**
mav_textDump, **245**
mav_textGetMatrix, **247**
mav_textGetSurfaceParams, **250**
mav_textGetUserdef, **251**
mav_textID, **253**
mav_texturedObjectsManage, 118, **340**
mav_texturedObjectsRender, 118, **340**
mav_timeGet, **209**
mav_timePrint, **125**
mav_timerPrint, **125**
mav_timerStart, 33, **209**
mav_timerStop, 33, **209**
mav_transparentObjectsManage, 119, **341**
mav_transparentObjectsRender, 119, 340, **341**
mav_transparentTextManage, 120, **342**, 343
mav_transparentTextRender, 120, **342**

mav_TRModuleID, **310**
mav_TRModuleInit, **311**

mav_vectorAdd, **210**
mav_vectorCrossProduct, **210**
mav_vectorDotProduct, **210**
mav_vectorMag, **210**, 211
mav_vectorMult, **210**, 211

mav_vectorMult3x3, **210**, 211
mav_vectorMult4x4, **210**, 211
mav_vectorNormalize, **210**, 211
mav_vectorPrint, **125**
mav_vectorRotate, **210**, 211
mav_vectorScalar, **210**, 211
mav_vectorScrnPos, **212**
mav_vectorSet, **210**
mav_vectorSub, **210**, 211
mav_vectorWorldPos, **213**
mav_viewParamsFixed, 36, **214**
mav_viewParamsPrint, **125**

mav_windowBackfaceCullGet, **215**
mav_windowBackfaceCullSet, **216**
mav_windowBackgroundColourSet, 139, 140, **217**
mav_windowBlendSet, **218**
mav_windowDelete, **219**
mav_windowDump, **220**
mav_windowFogSet, **221**
mav_windowLineStippleSet, **222**
mav_windowLineWidthSet, **223**
mav_windowMultiSampleSet, **224**
mav_windowNew, 219, **225**
mav_windowOrthogonalSet, **226**
mav_windowPaletteSet, **282**
mav_windowPerspectiveSet, **227**
mav_windowPolygonModeSet, **228**
mav_windowSet, **229**
mav_windowsModuleID, **310**
mav_windowsModuleInit, **311**
mav_windowViewModifierSet, 138, **230**
mav_windowViewParamsSet, 37, 139, 140, **232**

Types, Variables and Constants index

We use the following typographical conventions: **types** are in mixed case (example: MAV_matrix); **variables** are in lowercase (example: mav_class_any); and **constants** are in uppercase (example: MAV_ID_MATRIX).

MAP_LEAVE, 72
MAP_UNMAP, 79
MAV_180_OVER_PI, 349

MAV_2_PI, 349

MAV_ACCUM_ACCUM, 355
MAV_ACCUM_RETURN, 355
MAV_ANY_BUTTON, 352
MAV_avatar, 47

MAV_BACK, 355
MAV_BB, 5
MAV_BB_ACCURATE, 351
MAV_BB_FAST, 351
MAV_BLEND_1, 355
MAV_BLEND_OFF, 355
MAV_BLENDED_TEXTURE, 29, 208, 350
MAV_box, 7

MAV_callback, 65
MAV_callbackBBFn, 48
MAV_callbackCrossingFn, 49
MAV_callbackDeleteFn, 50
MAV_callbackDrawFn, 51
MAV_callbackDumpFn, 52
MAV_callbackExposeFn, 53
MAV_callbackFn, 54
MAV_callbackGetMatrixFn, 55
MAV_callbackGetSurfaceParamsFn, 56
MAV_callbackGetUserdefFn, 57
MAV_callbackIDFn, 58
MAV_callbackIntersectFn, 59
MAV_callbackKeyboardFn, 60
MAV_callbackMapFn, 61
MAV_callbackMouseFn, 62
MAV_callbackResizeFn, 63
MAV_callbackTDMFn, 64
MAV_CENTER_JUSTIFY, 351
MAV_CENTRE_JUSTIFY, 32
MAV_class, 66
MAV_clipPlane, 67
MAV_clipPlanes, 68
MAV_COLOUR, 29, 207, 350
MAV_colour, 69
MAV_COLOUR_BLACK, 350
MAV_COLOUR_BLUE, 350
MAV_COLOUR_GREEN, 350
MAV_COLOUR_RED, 350
MAV_COLOUR_WHITE, 350
MAV_composite, 8
MAV_compositeFormat, 70
MAV_compositeReadFn, 71
MAV_cone, 9
MAV_crossingEvent, 72
MAV_ctorus, 10
MAV_ctrlF, 112
MAV_cylinder, 11

MAV_DEG2RAD, 349
MAV_deviceCalcFn, 113
MAV_deviceEventFn, 114
MAV_devicePollFn, 115
MAV_DLISTS_COMPILE, 355
MAV_DLISTS_COMPILE_AND_EXECUTE, 355
MAV_DONTCARE, 347
MAV_drawInfo, 73

MAV_ellipse, 12
MAV_ENTER, 72, 352
MAV_EPSILON, 349

MAV_exposeEvent, 74

MAV_facet, 13
MAV_FALSE, 347
MAV_FILLED_FONT, 32, 351
MAV_font, 75
MAV_frameFn, 14
MAV_FRONT, 355

MAV_HBB, 39
MAV_hellipse, 15
MAV_hsphere, 16

MAV_ID_MATRIX, 349
MAV_ID_QUATERNION, 349
MAV_INFINITY, 349

MAV_KEY_ALT_L, 353
MAV_KEY_ALT_R, 353
MAV_KEY_CAPS_LOCK, 353
MAV_KEY_CTRL_L, 353
MAV_KEY_CTRL_R, 353
MAV_KEY_DOWN, 353
MAV_KEY_END, 353
MAV_KEY_F1, 353
MAV_KEY_F10, 353
MAV_KEY_F11, 353
MAV_KEY_F12, 353
MAV_KEY_F2, 353
MAV_KEY_F3, 353
MAV_KEY_F4, 353
MAV_KEY_F5, 353
MAV_KEY_F6, 353
MAV_KEY_F7, 353
MAV_KEY_F8, 353
MAV_KEY_F9, 353
MAV_KEY_HOME, 353
MAV_KEY_INSERT, 353
MAV_KEY_LEFT, 353
MAV_KEY_META_L, 353
MAV_KEY_META_R, 353
MAV_KEY_PAGE_DOWN, 353
MAV_KEY_PAGE_UP, 353
MAV_KEY_RIGHT, 353
MAV_KEY_SHIFT_L, 353
MAV_KEY_SHIFT_R, 353
MAV_KEY_UP, 353
MAV_keyboardEvent, 17, 84

MAV_LEAVE, 352
MAV_LEFT_BUTTON, 21, 352
MAV_LEFT_JUSTIFY, 32, 351
MAV_light, 76
MAV_LIGHT_ABSOLUTE, 186, 350
MAV_LIGHT_RELATIVE, 186, 350
MAV_lightingModel, 77
MAV_line, 19
MAV_list, 78
MAV_LIT_TEXTURE, 29, 207, 208, 350

MAV_MAP, 79, 352
MAV_mapEvent, 79
MAV_MATERIAL, 29, 207, 350
MAV_material, 80
MAV_matrix, 81
MAV_MATRIX_XAXIS_X, 349
MAV_MATRIX_XAXIS_Y, 349
MAV_MATRIX_XAXIS_Z, 349
MAV_MATRIX_XCOMP, 349
MAV_MATRIX_YAXIS_X, 349
MAV_MATRIX_YAXIS_Y, 349
MAV_MATRIX_YAXIS_Z, 349
MAV_MATRIX_YCOMP, 349
MAV_MATRIX_ZAXIS_X, 349
MAV_MATRIX_ZAXIS_Y, 349
MAV_MATRIX_ZAXIS_Z, 349
MAV_MATRIX_ZCOMP, 349
MAV_MAX_CBS, 347
MAV_MAX_CLIP_PLANES, 68, 347
MAV_MAX_WIN, 347
MAV_MIDDLE_BUTTON, 21, 352
MAV_MODELVIEW, 355
MAV_MODIFIER_ALT, 354
MAV_MODIFIER_CTRL, 354
MAV_MODIFIER_MAX, 354
MAV_MODIFIER_SHIFT, 354
MAV_moduleIDFn, 116
MAV_moduleInitFn, 117
MAV_mouseEvent, 20

MAV_navigatorFn, 82
MAV_NULL_VECTOR, 349

MAV_object, 85
MAV_objectIntersection, 84
MAV_objList, 83
mav_opt_delayTexture, 118

mav_opt_trans, 119, 120
MAV_OUTLINE_FONT, 32, 351

MAV_palette, 86
MAV_PI, 349
MAV_PI_OVER_180, 349
MAV_PI_OVER_2, 349
MAV_polygon, 23
MAV_POLYGON_FILL, 355
MAV_POLYGON_LINE, 355
MAV_polygonGrp, 22
MAV_polyline, 24
MAV_PRESSED, 352
MAV_PROJANDVIEW, 355
MAV_PROJECTION, 355
MAV_pyramid, 25

MAV_quaternion, 87

MAV_RAD2DEG, 349
MAV_rectangle, 26
MAV_REDEFINE_NOWARN, 347
MAV_REDEFINE_WARN, 347
MAV_RELEASED, 352
MAV_resizeEvent, 88
MAV_RIGHT_BUTTON, 21, 352
MAV_RIGHT_JUSTIFY, 32, 351
MAV_rtorus, 27

MAV_SILENT, 347
MAV_SMS, 111
MAV_SMSCallback, 41
MAV_SMSCallbackDeleteFn, 97
MAV_SMSCallbackEmptyFn, 98
MAV_SMSCallbackExecFnFn, 99
MAV_SMSCallbackFn, 100
MAV_SMSCallbackIntersectFn, 101
MAV_SMSCallbackObjectAddFn, 102
MAV_SMSCallbackObjectContainsFn, 103
MAV_SMSCallbackObjectNextFn, 104
MAV_SMSCallbackObjectRmvFn, 105
MAV_SMSCallbackPointerPopFn, 106
MAV_SMSCallbackPointerPushFn, 107
MAV_SMSCallbackPointerResetFn, 108
MAV_SMSCallbackSizeFn, 109
MAV_SMSClass, 110
MAV_SMSExecFn, 43
MAV_SMSExecFnFn, 42
MAV_SMSObj, 6
MAV_sphere, 28
MAV_STEREO_QUAD_BUFFERS, 347
MAV_STEREO_QUAD_BUFFERS_SEPARATE_Z, 347
MAV_STEREO_TWO_WINS, 347
MAV_STROKE_FONT, 32, 351
MAV_surfaceParams, 29

MAV_TDMCursor, 44
MAV_TDMEvent, 45
MAV_TDMPos, 46
MAV_teapot, 31
MAV_texCoord, 89
MAV_texEnvFn, 90
MAV_text, 32
MAV_TEXTURE, 29, 207, 350
MAV_texture, 91
MAV_texturedObjData, 118
MAV_THIS_VERSION, 347
MAV_time, 92
MAV_timer, 33
MAV_transObjData, 119
MAV_transTextData, 120
MAV_TRUE, 347

MAV_UNMAP, 352

MAV_vector, 34
MAV_VERBOSE, 347
MAV_VERSION, 347
MAV_viewModifierFn, 93
MAV_viewModifierParams, 35
MAV_viewParams, 36

MAV_WHEELDOWN_BUTTON, 21
MAV_WHEELUP_BUTTON, 21
MAV_window, 94

MAV_X_VECTOR, 349

MAV_Y_VECTOR, 349

MAV_Z_VECTOR, 349