RayCore® API Specifications

Version 1.0

SILICONARTS

Copyright© 2012 Siliconarts, Inc. All Rights Reserved.

This document is protected by copyright laws and contains proprietary materials to Siliconarts, Inc. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Siliconarts. You may use this specification for implementing the functionality therein, without altering or removing any trademark, copyright or other notice from the specification, but the receipt or possession of this specification does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Contents

Chapter 1 Introduction

1.1 3D Graphics

OpenGL ES is a software interface for graphics hardware in embedded system. The interface consists of a set of procedures and functions that allow a programmer to specify the objects and operations involved in producing high quality graphics images, especially color images of three-dimensional objects.

Many versions of OpenGL ES require that the graphics hardware contain a frame buffer. They are especially suitable for drawing objects such as points, lines and polygons; however, some functions in OpenGL ES are especially concerned with frame buffer manipulation. For example, some drawings including the operations such as antialiasing or texturing rely on a frame buffer.

OpenGL ES mainly supports raster based acceleration hardware. Raster based hardware in 3D graphics uses forward rendering which processes every polygon to generate images. Forward rendering is widely used, but its image quality is very low. To overcome the weakness, an additional hardware such as shader is implemented. As a result, the development period for application programs is long, and the development cost is high.

1.2 RayCore®

RayCore® is trademark of ray tracing rendering engine that is developed by Siliconarts, Inc. Unlike raster based hardward, RayCore® does not additional image processing but is capable of generating high quality 3D graphics images on real-time basis. Therefore, it is very easy to develop new 3D applications with RayCore®.

It supports the following major features:

- Backward rendering method
- Lighting
- Phong shading
- Shadow
- Reflection and refraction
- Texture mapping
- Antialiasing

In forward rendering to be used by raster based hardware generally, every polygon should be transformed to screen coordinates. This is because polygons will not be able to identify their position before rendering process is complete; hence depth test is required to secure the visibility of the polygons. On the other hand, backward rendering to be used by RayCore® can secure the visibility without depth test and also reduce data processing, since it does not process all of the polygons.

Lighting is set to display objects similar to those in real world, and hence produces various effects such as shadow, reflection and refraction. Texture mapping is a basic feature for photorealistic 3D graphics. Antialiasing also supports to provide higher image quality.

RayCore® renders 3D graphics images with a ray tracing algorithm containing an acceleration structure such as tree. A tree acceleration structure enables fast polygon searching during the process. A software tree builder is integreated into RayCore® API.

1.3 RayCore® API

RayCore® API is newly developed in OpenGL ES Version 1.1 - familiar format. New functions or parameters may be needed for ray tracing specific features. RayCore® API uses modified versions of the functions and parameters in OpenGL ES Version 1.1. Prefixes of the functions are changed, and the parameters are changed minimum. In addition, only a few functions are added for the ray tracing specific capabilities.

It supports the following major features:

- Vertex, normal, texture coordinate list
- Mipmap texture
- Triangle, strip, fan
- Material properties (ambient, diffuse, specular, reflection, refraction and etc.)
- Light properties (ambient, diffuse, specular and etc.)
- Matrix modes (Push, pop, load and etc.)

Chapter 2 Function List

This chapter describes RayCore® functions that are familiar to those in OpenGL ES 1.1. Most functions are similarly defined, while some of them are modified for RayCore®.

2.1 rcBindBuffer

void rcBindBuffer(RCenum *target*, RCuint *buffer*);	
Sets a buffer object with the name *buffer* to the current buffer target	
target	The current buffer target to which the buffer object is set (RC_ARRAY_BUFFER, RC_ELEMENT_ARRAY_BUFFER)
buffer	The name of the buffer object

Error Codes
RC_INVALID_ENUM : *target* is an invalid value **RC_OUT_OF_MEMORY : Failed to create the buffer object**

Related Functions
rcBufferData, rcBufferSubData, rcDeleteBuffers, rcGenBuffers rcGet RC_ARRAY_BUFFER_BINDING RC_ELEMENT_ARRAY_BUFFER_BINDING

Explanation

If the buffer object is set to the current buffer target, it is enabled to change and use its data. This buffer object remains active until the buffer object with a different name is set to the same buffer target, or until the buffer object is deleted. (See **rcDeleteBuffers**)

The buffer object name *buffer* is a non-negative integer, but the name that is actually used is a posivie integer. If the reserved value 0 is set to *buffer*, the settings of the buffer object are initialized to the current buffer target. If the buffer object with the corresponding name does not exist, this buffer object is automatically created. The buffer object with a new name can be generated using **rcGenBuffers**. Once created, the named buffer object may be set again to the current buffer target when needed.

When rcBindBuffer is called with the **RC_ARRAY_BUFFER**, target, vertex, normal or texture coordinate array pointer parameter of **rcDrawArrays**, which is commonly represented as a memory pointer, is instead interpreted as a buffer object managed in RayCore® API.

Also, when rcBindBuffer is called with the **RC_ELEMENT_ARRAY_BUFFER** target, the index array parameter of **rcDrawElements**, which is commonly represented as a memory pointer, is instead represented as a buffer object managed in RayCore® API.

2.2 rcBindMaterial

void rcBindMaterial(RCuint *material*);	
Sets a material object with the name *material* to the current material object	
material	The name of the material object

Error Codes
RC_OUT_OF_MEMORY : Failed to create the material object
Related Functions
rcDeleteMaterials, rcGenMaterials

Attention

rcBindMaterial must be required before a call to rcEnable, rcDisable, or rcBindTexture of RC_TEXTURE_2D or RC_TEXTURE_2D_NORMAL, and rcMaterial.

Explanation

If the material object is set to the current material object, it is enabled to change and use its data. This material object remains active until the material object with a different name is set to the current material object, or until this object is deleted. (See **rcDeleteMaterials**)

The material object name material is not a negative integer. The reserved value 0 represents the default material object name that is used to initialize the binding state of the current material object. If the material object with the corresponding name does not exist, this material object is automatically created. The material object with a new name can be generated using **rcGenMaterials**. Once created, the named material object may be set again to the current material object when needed.

2.3 rcBindTexture

void rcBindTexture(RCenum *target*, RCuint *texture*);	
Sets a texture object with the name *texture* to the current texture target	
target	The current texture target to which the texture object is set (RC_TEXTURE_2D, RC_TEXTURE_2D_NORMAL)
texture	The name of the textrue object

Error Codes

RC_INVALID_ENUM : *target* is an invalid value
RC_OUT_OF_MEMORY : Failed to create the texture object

Related Functions

rcDeleteTextures, rcGenTextures, rcTexImage2D, rcTexParameter
rcGet
RC_TEXTURE_BINDING_2D

Explanation

If the texture object is set to the current texture target, it is enabled to change and use its data. This texture object remains active until the texture object with a different name is set to the current texture target, or until the object is deleted. (See **rcDeleteTextures**) When the texture object is set, this object is applied to the current material object. (See **rcBindMaterial**)

The texture object name *texture* is not a negative integer. The reserved value 0 represents the default texture object name that is used to initialize the binding state of the current texture object. If the texture object with the corresponding name does not exist, this texture object is automatically created. The texture object with a new name can be generated using **rcGenTextures**. Once created, a named texture object may be set again to the current texture object when needed.

2.4 rcBufferData

void rcBufferData(RCenum *target*, RCsizeiptr *size*, const RCvoid * *data*, RCenum *usage*);

Sets the data information of a buffer object by initializing and creating the data store of the buffer object which is set to the current buffer target

target	The current buffer target to which the buffer data information is set (RC_ARRAY_BUFFER, RC_ELEMENT_ARRAY_BUFFER)
size	The size in bytes of the buffer object's new data store
data	The pointer of original data that will be copied into the data store of the buffer object (NULL if no data is to be copied)
usage	The usage pattern of the data store (RC_STATIC_DRAW, RC_DYNAMIC_DRAW)

Error Codes

RC_INVALID_ENUM
: *target* is an invalid value
: *usage* is an invalid value
RC_INVALID_VALUE : *size* is negative
RC_INVALID_OPERATION : The buffer ob ject name 0 is set to *target*
RC_OUT_OF_MEMORY : Failed to create the data store

Related Functions

rcBufferSubData, rcBindBuffer
rcGetBufferParameteriv
RC_BUFFER_SIZE
RCL_BUFFER_USAGE

Attention

If *data* is NULL, a data store of the specified size is still created, but its contents remain uninitialized and thus undefined.

Explanation

The previous data store for the current buffer object is deleted, and the new data store is created with the specified *size* in bytes. If the original data pointer *data* is NULL, the data store of the current buffer object is initialized.

2.5 rcBufferSubData

void rcBufferSubData(RCenum *target*, RCintptr *offset*, RCsizeiptr *size*, const RCvoid * *data*);

Replaces the sub or entire data information for the data store of a buffer object which is set to the current buffer target

target	The current buffer target to which the buffer data information is set (RC_ARRAY_BUFFER, RC_ELEMENT_ARRAY_BUFFER)
offset	The offset in bytes into the data store of the buffer object where data replacement will begin
size	The size in bytes of the data store region being replaced
data	The pointer to the new data that will be copied into the data store of the buffer object

Error Codes

RC_INVALID_ENUM : *target* is an invalid value
RC_INVALID_VALUE : *offset* or *size* is negative, or is beyond the region of the buffer object's allocated data store
RC_INVALID_OPERATION : The buffer object name 0 is set to *target*, or the buffer object *usage* is RC_STATIC_DRAW

Related Functions

rcBindBuffer, rcBufferData

Explanation

The data starting at byte *offset* and extending for *size* bytes is copied from the specified memory pointer *data* to the data store.

2.6 rcClearColor

void rcClearColor(RCclampf *red*, RCclampf *green*, RCclampf *blue*, RCclampf *alpha*);
void rcClearColorx(RCclampx *red*, RCclampx *green*, RCclampx *blue*, RCclampx *alpha*);

Sets the background color

red, green, blue, alpha	The red, green, blue, and alpha values of the background color

Related Functions

rcClear
rcGet
    RC_COLOR_CLEAR_VALUE

Explanation

rcClearColor specifies the red, green, blue, and alpha values used to set the background color. All initial values are 0. Values specified by **rcClearColor** are clamped to the range [0, 1].

2.7 rcColor

void rcColor4f(RCfloat *red*, RCfloat *green*, RCfloat *blue*, RCfloat *alpha*);
void rcColor4x(RCfixed *red*, RCfixed *green*, RCfixed *blue*, RCfixed *alpha*);
void rcColor4ub(RCubyte *red*, RCubyte *green*, RCubyte *blue*, RCubyte *alpha*);

Sets the default color

red, green, blue, alpha	The red, green, blue, and alpha values of the default color

Related Functions

rcBindMaterial, rcGenMaterial, rcMaterial
rcGet
    RC_CURRENT_COLOR

Attention

The initial value for the default color is (1, 1, 1, 1).

Explanation

In the current material properties, ambient and diffuse values are set with these RGBA values that are specified, and specular values are initialized to (0, 0, 0, 0).

Unsigned byte color components specified with **rcColor4ub** are linearly mapped to floating-point values such that 255 maps to 1.0 (full intensity), and 0 maps to 0.0 (zero intensity).

2.8 rcDeleteBuffers

void rcDeleteBuffers(RCsizei *n*, const RCuint * *buffers*);

Deletes buffer objects with *n* buffer object names in the array of buffer object names *buffers*

n	The count of buffer objects to be deleted
buffers	The array of buffer object names to be deleted

Error Codes

RC_INVALID_VALUE : *n* is negative

Related Functions

rcBufferData, rcBindBuffer, rcGenBuffers, rcIsBuffer

Explanation

When the buffer object is deleted, the reserved name 0 and the buffer object names that are yet to be created are ignored. The deleted buffer object has no contents, and its name is free for reuse. (See **rcGenBuffers**)

If the buffer object that is set to the buffer target is deleted, all bindings to that object are reset to 0.

2.9 rcDeleteMaterials

void rcDeleteMaterials(RCsizei *n*, const RCuint * *materials*);

Deletes material objects with *n* material object names in the array of material object names *materials*

n	The count of material objects to be deleted
materials	The array of material object names to be deleted

Error Codes

RC_INVALID_VALUE : *n* is negative

Related Functions

rcBindMaterial, rcGenMaterials, rcIsMaterial

Explanation

When the material object is deleted, the reserved name 0 and the material object names that are yet to be created are ignored. The deleted material object has no contents, and its name is free for reuse. (See **rcGenMaterials**)

If the material object that is set to the current material object is deleted, the current material object reverts to 0 (the default material).

2.10 rcDeleteTextures

void rcDeleteTextures(RCsizei *n*, const RCuint * *textures*);

Deletes texture objects with *n* texture object names in the array of texture object names *textures*

n	The count of texture objects to be deleted
textures	The array of texture object names to be deleted

Error Codes

RC_INVALID_VALUE : *n* is negative

Related Functions

rcBindTexture, rcGenTextures, rcIsTexture

Explanation

When the texture objects are deleted, the reserved name 0 and the texture object names that are yet to be created are ignored. The deleted texture objects have no contents, and its name is free for reuse. (See **rcGenTextures**)

If the texture object that is set to the current texture object is deleted, the current texture object reverts to 0 (the default texture).

2.11 rcDrawArrays

void rcDrawArrays(RCenum *mode*, RCint *first*, RCsizei *count*);	
Creates primitives to render from the array of preset vertices, normals, colors, and texture coordinates	
mode	The kind of primitives to render (RC_TRIANGLE_STRIP, RC_TRIANGLE_FAN, RC_TRIANGLES, RC_QUADS)
first	The starting index in the enabled arrays
count	The count of indices to be used in creating primitives

Error Codes
RC_INVALID_ENUM : *mode* is an invalid value **RC_INVALID_VALUE : *count* is negative**
Related Functions
rcDrawElements, rcNormalPointer, rcTexCoordPointer, rcVertexPointer

Explanation

After prespecifying the separate arrays of vertices, normals, colors, and texture coordinates, primitives to render can be constructed by calling **rcDrawArrays**.

Geometric primitives are constructed by using *count* sequential elements from the starting index, *first*, of each array. *mode* means what kind of primitives is constructed, and how those primitives are constructed by the array elements. If the vertex array is not enabled, no geometric primitives are created.

2.12 rcDrawElements

void rcDrawElements(RCenum *mode*, RCsizei *count*, RCenum *type*, const RCvoid * *indices*);

Creates primitives to render from arrays of preset vertices, normals, colors, texture coordinates and indices

mode	The kind of primitives to render (RC_TRIANGLE_STRIP, RC_TRIANGLE_FAN, RC_TRIANGLES, RC_QUADS)
count	The count of indices to be used in creating primitives
type	The data type of an indices array (RC_BYTE, RC_UNSIGNED_BYTE, RC_SHORT, RC_UNSIGNED_SHORT, RC_INT, RC_UNSIGNED_INT)
indices	The pointer to data store where the indices are loaded

Error Codes

RC_INVALID_ENUM
: *mode* is an invalid value
: *type* is an invalid value
RC_INVALID_VALUE : *count* is negative

Related Functions

rcDrawArrays, rcNormalPointer, rcTexCoordPointer, rcVertexPointer

Explanation

After prespecifying the separate arrays of vertices, normals, colors, texture coordinates, and related indices, primitives to render can be constructed by calling **rcDrawElements**.

Geometric primitives are constructed by using *count* sequential indices from the pointer, *indices*, of an indices array to lookup elements in each array. *mode* means what kind of primitives is constructed, and how those primitives are constructed with with the array elements. If the vertex array is not enabled, no geometric primitives are created.

2.13 rcEnable

void rcEnable(RCenum *cap*);
void rcDisable(RCenum *cap*);

Enables or disables various capabilities of RayCore® API

cap	The capability of RayCore® API

Error Codes

RC_INVALID_ENUM : *cap* is an invalid value

Related Functions

rcEnableClientState, rcGet, rcIsEnabled, rcLight, rcMaterial, rcTexImage2D, rcTexSubImage2D

Explanation

The following is the description of RayCore® API capability *cap*.

- RC_LIGHTi
  Enables or disables the ith light source. (See **rcLight**)
- RC_LIGHTING
  Enables or disables lighting. (See **rcLight**)
- RC_MATRIX_PALETTE_OES
  Enables or disables palette matrix. (See **rcCurrentPaletteMatrixOES** and **rcLoadPaletteFromModelViewMatrixOES**)
- RC_TEXTURE_2D
  Enables or disables two-dimensional texturing which is performed for the active texture unit and the current material. (See **rcBindMaterial** and **rcTexImage2D**)
- RC_TEXTURE_2D_NORMAL
  Enables or disables two-dimensional normal map texturing which is performed for the active texture unit and the current material. (See **rcBindMaterial** and **rcTexImage2D**)
- RC_USE_COLOR_SHADOW
  Sets whether color values of each material object are applied to shadow. If it is disabled, grayscale colors are simply applied. (See **rcMaterial** and **rcLight**)
- RC_USE_SHADOW
  Sets whether shadow is rendered. If it is disabled, no shadow is rendered.
- RC_USE_TEXTURE_ALPHA_SHADOW
  Sets whether alpha values of a material texture is applied to shadow. If it is disabled, the texture without alpha values is simply applied. (See **rcMaterial** and **rcTexImage2D**)

- RC_USE_TEXTURE_ONLY:
  Sets whether shadow and reflection effects are applied to the texture of the current material object. If it is disabled, shadow and reflection effects are applied. (See **rcMaterial** and **rcTexImage2D**)

- RC_USE_TRANSMITTANCE_SHADOW
  Sets whether the transmittance value of a material object is applied to shadow. If it is disabled, the dark shadow is generated without the transmittance effect. (See **rcMaterial** and **rcTexImage2D**)

2.14 rcEnableClientState

void rcEnableClientState(RCenum *array*);
void rcDisableClientState(RCenum *array*);

Enables or disables the individual client state

array	The client state (RC_COLOR_ARRAY, RC_MATRIX_INDEX_ARRAY_OES, RC_NORMAL_ARRAY, RC_TEXTURE_COORD_ARRAY, RC_VERTEX_ARRAY, and RC_WEIGHT_ARRAY_OES)

Error Codes

RC_INVALID_ENUM : *array* is an invalide value

Related Functions

rcDrawArrays, rcDrawElements, rcEnable, rcIsEnabled, rcNormalPointer, rcTexCoordPointer, rcVertexPointer, rcMatrixIndexPointerOES, rcWeightPointerOES

Explanation

By default, all client states are disabled. The following is the description of the client state *array*.

- RC_MATRIX_INDEX_ARRAY_OES
  Enables or disables the palette matrix index array to use during rendering when **rcDrawArrays** or **rcDrawElements** is called. (See **rcMatrixIndexPointerOES**)

- RC_NORMAL_ARRAY
  Enables or disables the normal array to be used during rendering when **rcDrawArrays** or **rcDrawElements** is called. (See **rcNormalPointer**)

- RC_TEXTURE_COORD_ARRAY
  Enables or disables the texture coordinate array to be used during rendering when **rcDrawArrays** or **rcDrawElements** is called. (See **rcTexCoordPointer**)

- RC_VERTEX_ARRAY
  Enables or disables the vertex array to be used during rendering when **rcDrawArrays** or **rcDrawElements** is called. (See **rcVertexPointer**)

- RC_WEIGHT_ARRAY_OES
  Enables or disables the weight array to use during rendering when **rcDrawArrays** or **rcDrawElements** is called. (See **rcWeightPointerOES**)

2.15 rcFinish

void rcFinish(void);
Executes ray tracing rendering

Related Functions
rcFlush, eglSwapBuffers

Explanation

It executes ray tracing rendering by RayCore®.

2.16 rcFlush

void rcFlush(void);
Executes the same operation with rcFinish

Related Functions
rcFinish

Explanation

rcFlush calls **rcFinish**.

2.17 rcFrustum

void rcFrustumf(RCfloat *left*, RCfloat *right*, RCfloat *bottom*, RCfloat *top*, RCfloat *near*, RCfloat *far*);
void rcFrustumx(RCfixed *left*, RCfixed *right*, RCfixed *bottom*, RCfixed *top*, RCfixed *near*, RCfixed *far*);

Sets a viewing frustum into the world coordinate system

left, right	The coordinates for the left and right vertical clipping planes
bottom, top	The coordinates for the bottom and top horizontal clipping planes
near, far	The distances to the near and far depth clipping planes (Both distances must be positive.)

Error Codes

RC_INVALID_VALUE
: *near* or *far* is not positive
: *near* is equal to *far*
: *left* is equal to *right*, or *bottom* is equal to *top*

Related Functions

rcViewport

Explanation

The starting position of ray generation, the pixel size of clipping planes, and the distance of camera and clipping planes are calculated with these specified values. *zNear* is the distance from the camera to the screen. *zFar* is not used.

2.18 rcGenBuffers

void rcGenBuffers(RCsizei *n*, RCuint * *buffers*);

Generates new buffer objects with *n* buffer object names in the array of buffer object names *buffers*

n	The count of buffer objects to be generated
buffers	The array in which the generated buffer object names are stored

Error Codes

RC_INVALID_VALUE : *n* is negative
RC_OUT_OF_MEMORY : Failed to create the buffer object

Related Functions

rcBindBuffer, rcBufferData, rcBufferSubData, rcDeleteBuffers, rcIsBuffer

Explanation

Returned names of the buffer objects, which are yet to be either created or deleted, are not always continuous integers. This is because the buffer objects that have already been generated are not be returned by calling **rcGenBuffers** unless they are deleted by calling **rcDeleteBuffers**.

2.19 rcGenMaterials

void rcGenMaterials(RCsizei *n*, RCuint * *materials*);

Generates new material objects with *n* material object names in the array of material object names *materials*

n	The count of material objects to be generated
materials	The array in which the generated material object names are stored

Error Codes

RC_INVALID_VALUE : *n* is negative
RC_OUT_OF_MEMORY : Failed to create the material object

Related Functions

rcBindMaterial, rcDeleteMaterials, rcIsMaterial

Explanation

Returned names of the mateial objects, which are yet to be either created or deleted, are not always continuous integers. This is because the material objects that have already been generated are not returned by calling **rcGenMaterials** unless they are first deleted by calling **rcDeleteMaterials**.

2.20 rcGenTextures

void rcGenTextures(RCsizei *n*, RCuint * *textures*);

Generates new texture objects with *n* texture object names in the array of texture object names *textures*

n	The count of texture objects to be generated
textures	The array in which the generated texture object names are stored

Error Codes

RC_INVALID_VALUE : *n* is negative
RC_OUT_OF_MEMORY : Failed to create the texture object

Related Functions

rcBindTexture, rcDeleteTextures, rcIsTexture, rcTexImage2D, rcTexParameter

Explanation

Returned names of the texture objects, which are yet to be either created or deleted, are not always continuous integers. This is because the texture objects that have already been generated are not returned by calling **rcGenTextures** unless they are first deleted by calling **rcDeleteTextures**.

2.21 rcGet

void rcGetBooleanv(RCenum *pname*, RCboolean * *params*);
void rcGetFixedv(RCenum *pname*, RCfixed * *params*);
void rcGetFloatv(RCenum *pname*, RCfloat * *params*);
void rcGetIntegerv(RCenum *pname*, RCint * *params*);

Returns the values of static state variables of RayCore® API

pname	The parameter value of static state variables to be returned
params	The pointer to an array of data to be returned

Error Codes

RC_INVALID_ENUM : *pname* is an invalid value

Related Functions

rcGetError, rcGetString, rcCurrentPaletteMatrixOES, rcLoadPaletteFromModelViewMatrixOES, rcMatrixIndexPointerOES, rcWeightPointerOES

Explanation

If the returned value type is different from the type of the value being obtained, a type conversion is performed.

- GetBooleanv
  A floating-point or integer value is converted to **RC_FALSE** if and only if it is 0, otherwise it converts to **RC_TRUE**.

- GetIntegerv
  If the value is not an RGBA color component, a boolean value is converted to either 1 or 0, and a floating-point value is rounded to the nearest integer. Otherwise it performs a linear mapping that maps a floating-point value 1.0 to the integer value 255, and a floating-point value 0.0 to the integer value 0.

- GetFloatv
  A boolean value is converted to either 1.0 or 0.0

The following is the description of the static state variable *pname*.

- RC_ARRAY_BUFFER_BINDING
  Returns the name of the buffer object currently specified to the target **RC_ARRAY_BUFFER**. If no buffer object is set to this target, 0 is returned. (See **rcBindBuffer**)

- RC_COLOR_CLEAR_VALUE
  Returns the red, green, blue, and alpha values of the background color. (See **rcClearColor**)

- RC_CURRENT_COLOR
  Returns the red, green, blue, and alpha values of the default color. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the integer value 255, and 0.0 returns the integer value 0. (See **rcColor**)

- RC_ELEMENT_ARRAY_BUFFER_BINDING
  Returns the name of the buffer object currently specified to the target **RC_ELEMENT_ARRAY_BUFFER**. If no buffer object is set to this target, 0 is returned. (See **rcBindBuffer**)

- RC_LIGHTi
  Returns the active state of the ith light source. (See **rcLight**)

- RC_LIGHTING
  Returns the active state of the lighting. (See **rcLight** and **rcMaterial**)

- RC_MAX_LIGHTS
  Returns the maximum number of the light sources. This value is 8. (See **rcLight**)

- RC_MAX_MODELVIEW_STACK_DEPTH
  Returns the maximum supported depth of the modelview matrix stack. The value is 32. (See **rcPushMatrix**)

- RC_MAX_PALETTE_MATRICES_OES
  Returns the maximum number of the palette matrix. This value is 128. (See **rcCurrentPaletteMatrixOES**)

- RC_MAX_PROJECTION_STACK_DEPTH
  Returns the maximum supported depth of the projection matrix stack. The value is 32. (See **rcPushMatrix**)

- RC_MAX_TEXTURE_SIZE
  Returns the maximum size of the texture supported in RayCore®. This value is 1024. (See **rcTexImage2D**)

- RC_MAX_TEXTURE_STACK_DEPTH
  Returns the maximum supported depth of the texture matrix stack. The value is 32. (See **rcPushMatrix**)

- RC_MAX_TEXTURE_UNITS
  Returns the count of texture units supported in RayCore®. This value is 1. RayCore® API does not support the multi texture.

- RC_MAX_THRESHOLD_LEVELS

Returns the maximum number of the ray bounce threshold level. The value is 10. (See **rcHint**)

- RC_MAX_VERTEX_UNITS_OES
  Returns the maximum number of the vertex units for the palette matrix. The value is 128. (See **rcMatrixIndexPointerOES** and **rcWeightPointerOES**)

- RC_MAX_VIEWPORT_DIMS
  Returns the maximum supported width and height of the viewport. The value is 2048. (See **rcViewport**)

- RC_MATRIX_INDEX_ARRAY_OES
  Returns the active state of the palette matrix index array. (See **rcMatrixIndexPointerOES**)

- RC_MATRIX_INDEX_ARRAY_BUFFER_BINDING_OES
  Returns the name of the buffer object specified to the palette matrix index array. (See **rcMatrixIndexPointerOES**)

- RC_MATRIX_INDEX_ARRAY_SIZE_OES
  Returns the count of matrix indices per vertex in the palette matrix index array. (See **rcMatrixIndexPointerOES**)

- RC_MATRIX_INDEX_ARRAY_STRIDE_OES
  Returns the byte length between two adjacent matrix indices in the palette matrix index array. (See **rcMatrixIndexPointerOES**)

- RC_MATRIX_INDEX_ARRAY_TYPE_OES
  Returns the data type of the palette matrix index array. (See **rcMatrixIndexPointerOES**)

- RC_MATRIX_MODE
  Returns the current matrix mode. (See **rcMatrixMode**)

- RC_MATRIX_PALETTE_OES
  Returns the active state of the palette matrix. (See **rcEnable** and **rcDisable**)

- RC_MODELVIEW_MATRIX
  Returns 16 values of the current modelview matrix. (See **rcPushMatrix**)

- RC_MODELVIEW_STACK_DEPTH
  Returns the count of matrices on the modelview matrix stack. (See **rcPushMatrix**)

- RC_NORMAL_ARRAY
  Returns the active state of the normal array. (See **rcNormalPointer**)

- RC_NORMAL_ARRAY_BUFFER_BINDING
  Returns the name of the buffer object specified to the normal array. (See **rcNormalPointer**)

- RC_NORMAL_ARRAY_STRIDE
  Returns the byte length between two adjacent normals in the normal array. (See **rcNormalPointer**)

- RC_NORMAL_ARRAY_TYPE
  Returns the data type of the normal array. (See **rcNormalPointer**)

- RC_PROJECTION_MATRIX
  Returns 16 values of the current projection matrix. (See **rcPushMatrix**)

- RC_PROJECTION_STACK_DEPTH
  Returns the count of matrices on the projection matrix stack. (See **rcPushMatrix**)

- RC_TEXTURE_2D
  Returns the active state of the 2D texturing. (See **rcTexImage2D**)

- RC_TEXTURE_2D_NORMAL
  Returns the active state of the 2D normal map texturing. (See **rcTexImage2D**)

- RC_TEXTURE_BINDING_2D
  Returns the name of the texture object currently specified to the current texture target (**RC_TEXTURE_2D or RC_TEXTURE_2D_NORMAL)**. (See **rcBindTexture**)

- RC_TEXTURE_COORD_ARRAY
  Returns the active state of the texture coordinate array. (See **rcTexCoordPointer**)

- RC_TEXTURE_COORD_ARRAY_BUFFER_BINDING
  Returns the name of the buffer object specified to the texture coordinate array. (See **rcTexCoordPointer**)

- RC_TEXTURE_COORD_ARRAY_SIZE
  Returns the count of coordinates per element in the texture coordinate array. (See **rcTexCoordPointer**)

- RC_TEXTURE_COORD_ARRAY_STRIDE
  Returns the byte length between two adjacent elements in the texture coordinate array. (See **rcTexCoordPointer**)
- RC_TEXTURE_COORD_ARRAY_TYPE
  Returns the data type of a texture coordinate array. (See **rcTexCoordPointer**)

- RC_TEXTURE_MATRIX
  Returns 16 values of the current texture matrix. (See **rcPushMatrix**)

- RC_TEXTURE_STACK_DEPTH
  Returns the count of matrices on the texture matrix stack. (See **rcPushMatrix**)

- RC_USE_COLOR_SHADOW
  Returns the active state of the color shadow. (See **rcEnable** and **rcDisable**)

- RC_USE_SHADOW
  Returns the active state of the shadow. (See **rcEnable** and **rcDisable**)

- RC_USE_TEXTURE_ALPHA_SHADOW
  Returns the active state of the alpha texture shadow. (See **rcEnable** and **rcDisable**)

- RC_USE_TEXTURE_ONLY
  Returns the active state of the background texture. (See **rcEnable** and **rcDisable**)

- RC_USE_TRANSMITTANCE_SHADOW
  Returns the active state of the transmittance shadow. (See **rcEnable** and **rcDisable**)

- RC_VIEWPORT
  Returns the x and y window coordinates of the viewport, followed by its width and height. (See **rcViewport**)

- RC_VERTEX_ARRAY
  Retuns the active state of the vertex array. (See **rcVertexPointer**)

- RC_VERTEX_ARRAY_BUFFER_BINDING
  Returns the name of the buffer object specified to the vertex array. (See **rcVertexPointer**)

- RC_VERTEX_ARRAY_SIZE
  Returns the count of coordinates per vertex in the vertex array. (See **rcVertexPointer**)

- RC_VERTEX_ARRAY_STRIDE
  Returns the byte length between two adjacent vertices in the vertex array. (See **rcVertexPointer**)

- RC_VERTEX_ARRAY_TYPE
  Returns the data type of the vertex array. (See **rcVertexPointer**)

- RC_WEIGHT_ARRAY_OES
  Returns the active state of the palette matrix weight array. (See **rcWeightPointerOES**)
- RC_WEIGHT_ARRAY_BUFFER_BINDING_OES
  Returns the name of the buffer object specified to the palette matrix weight array. (See **rcWeightPointerOES**)

- RC_WEIGHT_ARRAY_SIZE_OES
  Returns the count of weights per vertex in the palette matrix weight array. (See **rcWeightPointerOES**)

- RC_WEIGHT_ARRAY_STRIDE_OES
  Returns the byte length between two adjacent weights in the palette matrix weight array. (See **rcWeightPointerOES**)

- RC_WEIGHT_ARRAY_TYPE_OES
  Returns the data type of the palette matrix weight array. (See **rcWeightPointerOES**)

2.22 rcGetBufferParameteriv

void rcGetBufferParameteriv(RCenum *target*, RCenum *pname*, RCint * *params*);	
Returns the information of a buffer object that is set to the current buffer target	
target	The current buffer target to request for information (RC_ARRAY_BUFFER, RC_ELEMENT_ARRAY_BUFFER)
pname	The information parameter of the buffer object (RC_BUFFER_SIZE, RC_BUFFER_USAGE)
params	The pointer to an array of data to be returned

Error Codes
RC_INVALID_ENUM : *pname* is an invalid value **RC_INVALID_OPERATION : The buffer object name 0 is set to *target***
Related Functions
rcBufferData, rcBindBuffer

Explanation

The following is the description of the returned value on the information parameter of a buffer object *pname*.

- RC_BUFFER_SIZE
  Returns the size in bytes of the buffer object.

- RC_BUFFER_USAGE
  Returns the usage pattern of the buffer object.

2.23 rcGetError

RCenum rcGetError(void);
Returns the error code that is currently set

Explanation

Initially, the error flag is set to **RC_NO_ERROR**. When an error occurs during RayCore® operation, the error flag is set to the appropriate error code value.

No other errors are recorded until **rcGetError** is called. After the error code is returned, the error flag is reset to **RC_NO_ERROR**. If **rcGetError** returns **RC_NO_ERROR**, there has been no detectable error since the last call to **rcGetError**, or since RayCore® API was initialized. The following is the definition of errors.

- RC_NO_ERROR
  No error occurs. This value is 0.
- RC_INVALID_ENUM
  An unacceptable value is specified for an enumerated argument.
- RC_INVALID_VALUE
  A numeric argument is out of range.
- RC_INVALID_OPERATION
  The specified operation is not allowed in the current state.
- RC_STACK_OVERFLOW
  This command would cause a stack overflow.
- RC_STACK_UNDERFLOW
  This command would cause a stack underflow.
- RC_OUT_OF_MEMORY
  There is not enough memory left to execute the command.

2.24 rcGetLight

void rcGetLightfv(RCenum *light*, RCenum *pname*, RCfloat * *params*);
void rcGetLightxv(RCenum *light*, RCenum *pname*, RCfixed * *params*);

Returns the information of a light source

light	The light source to request for information, RC_LIGHT*i* (0<= *i* < RC_MAX_LIGHTS) (The maximum number of supported light source is 8.)
pname	The information parameter of the light source (RC_AMBIENT, RC_DIFFUSE, RC_SPECULAR, RC_POSITION, RC_SPOT_DIRECTION, RC_SPOT_EXPONENT, RC_SPOT_INNER_CONE, RC_SPOT_OUTER_CONE, RC_SPOT_CUTOFF, RC_ATTENUATION_RANGE, RC_START_ATTENUATION¸ RC_END_ATTENUATION, RC_CONSTANT_ATTENUATION, RC_LINEAR_ATTENUATION, RC_QUADRATIC_ATTENUATION)
params	The pointer to an array of data to be returned

Error Codes

RC_INVALID_ENUM : *light* or *pname* is an invalide value

Related Functions

rcLight

Attention

RC_LIGHT*i* and RC_LIGHT0+*i* are the same.

Explanation

The following is the description of the returned value on the information parameter of a light source *pname*.

- RC_AMBIENT
  Returns the ambient RGBA intensity of the light source.
- RC_DIFFUSE
  Returns the diffuse RGBA intensity of the light source.
- RC_SPECULAR
  Returns the specular RGBA intensity of the light source.
- RC_POSITION

Returns 4 values representing the position and type of the light source. First 3 values are coordinates of the position in the eye coordinate system. Last value is the type (0 : directional light, 1 : point light) of the light source.

- RC_SPOT_DIRECTION
  Returns 3 values representing the direction of the light source.

- RC_SPOT_EXPONENT
  Returns the spot exponent of the light source that is related to the intensity distribution.

- RC_SPOT_INNER_CONE
  Returns the attenuated start angle of the light source that is related to the intensity distribution.

- RC_SPOT_OUTER_CONE, RC_SPOT_CUTOFF
  Returns the maximum spread angle of the light source that is related to the intensity distribution.

- RC_ATTENUATION_RANGE
  Returns the distance range from the light position that is attenuated. First value is the start distance, second value is the end distance.

- RC_START_ATTENUATION
  Returns the distance from the light source to begin the intensity attenuation.

- RC_END_ATTENUATION
  Returns the distance from the light source to end the intensity attenuation.

- RC_CONSTANT_ATTENUATION, RC_LINEAR_ATTENUATION, RC_QUADRATIC_ATTENUATION
  Returns one of the three attenuation coefficient on the intensity of the light source.

2.25 rcGetMaterial

void rcGetMaterialfv(RCenum *face*, RCenum pname, RCfloat * *params*); **void rcGetMaterialxv(RCenum *face*, RCenum *pname*, RCfixed * *params*);**	
Returns the information of a material object	
face	The face of a current material object to request for information (RC_FRONT, RC_BACK, RC_FRONT_AND_BACK)
pname	The information parameter of the material object (RC_AMBIENT, RC_DIFFUSE, RC_SHININESS, RC_SPECULAR, RC_SPECULAR_LEVEL, RC_REFLECTION, RC_TRANSMITTANCE, RC_REFRACTION_INDEX)
params	The pointer to an array of data to be returned

Error Codes
RC_INVALID_ENUM : *face* or *pname* is an invalide value
Related Functions
rcBindMaterial, rcGenMaterial, rcMaterial

Attention

In RayCore® API, there is only one material shared by the front and back. Therefore querying **RC_FRONT**, querying **RC_BACK**, and querying **RC_RONT_AND_BACK** will always return the same value.

Explanation

The following is the description of the returned value on the information parameter of the material object *pname*.

- RC_AMBIENT
  Returns the ambient RGBA reflectance of the material object.
- RC_DIFFUSE
  Returns the diffuse RGBA reflectance of the material object.
- RC_REFLECTION
  Returns the reflectance of the material object.
- RC_REFRACTION_INDEX
  Returns the refraction index of the material object.

- RC_SHININESS
  Returns the specular exponent of the material object.

- RC_SPECULAR
  Returns the specular RGBA reflectance of the material object.

- RC_SPECULAR_LEVEL
  Returns the specular intensity of the material object.

- RC_TRANSMITTANCE
  Returns the transmittance of the material object.

2.26 rcGetPointerv

void rcGetPointerv(RCenum *pname*, RCvoid ** *params*);	
Returns the memory address of a pointer to the current array	
pname	The type of the pointer to the current array (RC_COLOR_ARRAY_POINTER, RC_MATRIX_INDEX_ARRAY_POINTER_OES , RC_NORMAL_ARRAY_POINTER, RC_TEXTURE_COORD_ARRAY_POINTER, RC_VERTEX_ARRAY_POINTER, RC_WEIGHT_ARRAY_POINTER_OES)
params	The memory address of the pointer specified by *pname*

Error Codes
RC_INVALID_ENUM : *pname* is an invalid value
Related Functions
rcBindBuffer, rcDrawArrays, rcMatrixIndexPointerOES, rcNormalPointer, rcTexCoordPointer, rcVertexPointer, rcWeightPointerOES

2.27 rcGetString

const RCubyte * rcGetString(RCenum *name*);	
Returns a string describing RayCore® API	
name	The information type (RC_VENDOR, RC_RENDERER, RC_VERSION)

Error Codes
RC_INVALID_ENUM : *name* is an invalid value

Attention

If an error occurs, **rcGetString** returns **NULL**.

Explanation

All strings are null-terminated. The following is the description of the returned string on the information type *name*.

- RC_VENDOR
  Returns the name of the company that implements RayCore® API.

- RC_RENDERER
  Returns the name of the renderer.

- RC_VERSION
  Returns the version number. The form of this string is "RayCore API <major>.<minor>", where <major> and <minor> are integers. "RayCore API 1.0" will have 1 for <major> and 0 for <minor>.

2.28 rcGetTexParameter

void rcGetTexParameterfv(RCenum *target*, RCenum *pname*, RCfloat * *params*);
void rcGetTexParameteriv(RCenum *target*, RCenum *pname*, RCint * *params*);
void rcGetTexParameterxv(RCenum *target*, RCenum *pname*, RCfixed * *params*);

Returns the information of the texture target

target	The texture target to request for information (RC_TEXTURE_2D, RC_TEXTURE_2D_NORMAL)
pname	The information parameter of the texture target (RC_TEXTURE_WRAP_S, RC_TEXTURE_WRAP_T, RC_GENERATE_MIPMAP)
params	The pointer to an array of data to be returned

Error Codes

RC_INVALID_ENUM : *target* or *pname* is an invalid value

Related Functions

rcTexParameter

Explanation

The following is the description of a returned value on the information parameter of the texture target *pname*.

- RC_TEXTURE_WRAP_S
  Returns the information of the wrapping function for texture coordinate *s*.
  (**RC_CLAMP_TO_EDGE, RC_REPEAT**)

- RC_TEXTURE_WRAP_T
  Returns the information of the wrapping function for texture coordinate *t*.
  (**RC_CLAMP_TO_EDGE, RC_REPEAT**)

- RC_GENERATE_MIPMAP
  Returns the active state of the automatic mipmap level update. (See **rcTexParameter**)

2.29 rcHint

void rcHint(RCenum *target*, RCenum *mode*);	
Sets the hint for a behavior to be controlled	
target	The type of the behavior to be controlled (RC_RENDERING_HINT, RC_RAYBOUNCE_THRESHOLD_HINT, RC_MIPMAP_HINT)
mode	The state of the desired behavior (RC_FASTEST, RC_NICEST, RC_FASTEST_AND_NICEST, RC_THRESHOLD_LEVELi, RC_DONT_CARE) RC_THRESHOLD_LEVEL*i* (0<= *i* < RC_MAX_THRESHOLD_LEVELS)

Error Codes
RC_INVALID_ENUM : *target* or *mode* is an invalid value

Attention

RC_THRESHOLD_LEVEL*i* and RC_THRESHOLD_LEVEL0+*i* are the same.

Explanation

Some behavior can be controlled with hints in RayCore® API. Following is a description of the desired behavior state, *mode*.

- RC_FASTEST
  Selects the state of the most efficient behavior.
- RC_NICEST
  Selects the state of the behavior that represents the most highest quality.
- RC_FASTEST_AND_NICEST
  Selects the state of the behavior that represents the most efficient and highest quality.
- RC_DONT_CARE
  Selects the default state of the behavior.

The initial value for each *target* is **RC_DONT_CARE**. The following is the interpretation of the behavior state, *mode*, by each behavior type *target*.

- RC_RENDERING_HINT

Indicates the rendering quality of the pixel sampling. Hinting **RC_DONT_CARE**, **RC_FASTEST**, **RC_NICEST**, **or RC_FASTEST_AND_NICEST** can result in per-pixel rendering of sampling effects.

- RC_RAYBOUNCE_THRESHOLD_HINT

  Indicates the threshold level of ray bounce for pixel value. If a threshold level is not applied, hinting **RC_DONE_CARE**. Ortherwise hinting **RC_THRESHOLD_LEVEL***i* can result in per-pixel rendering of the threshold level effects.

- RC_MIPMAP_HINT

  Indicates the texture quality of rendering the material object. Hinting **RC_DONT_CARE**, **RC_FASTEST**, or RC_**NICEST** can result in using the mipmap level per material texture.

2.30 rcIsBuffer

RCboolean rcIsBuffer(RCuint *buffer*);	
Returns if a buffer object that has been created with the name, *buffer*, is present	
buffer	The name of the buffer object

Related Functions
rcBindBuffer, rcBufferData, rcBufferSubData, rcDeleteBuffers, rcGenBuffers

Explanation

Returns **RC_TRUE** if *buffer* is currently the name of the buffer object that has been created. Ortherwise, returns **RC_FALSE** when *buffer* is non-zero.

2.31 rcIsEnabled

RCboolean rcIsEnabled(RCenum *cap*);	
Returns the active state for the various capabilities of RayCore® API	
cap	The capability of RayCore® API

Error Codes
RC_INVALID_ENUM : *cap* is an invalid value

Error Codes

rcEnable, rcEnableClientState

Attention

If an error occurs, **rcIsEnabled** returns 0.

Explanation

The following is the description of RayCore® API capability *cap*.

Capability	**See function:**
RC_LIGHT*i*	rcLight
RC_LIGHTING	rcLight, rcMaterial
RC_MATRIX_INDEX_ARRAY_OES	rcEnableClientState
RC_MATRIX_PALETTE_OES	rcEnable
RC_NORMAL_ARRAY	rcNormalPointer
RC_TEXTURE_2D	rcEnable, rcTexImage2D, rcMaterial
RC_TEXTURE_2D_NORMAL	rcEnable, rcTexImage2D, rcMaterial
RC_TEXTURE_COORD_ARRAY	rcTexCoordPointer
RC_USE_COLOR_SHADOW	rcEnable
RC_USE_SHADOW	rcEnable
RC_USE_TEXTURE_ALPHA_SHADOW	rcEnable
RC_USE_TEXTURE_ONLY	rcEnable
RC_USE_TRANSMITTANCE_SHADOW	rcEnable
RC_VERTEX_ARRAY	rcVertexPointer
RC_WEIGHT_ARRAY_OES	rcEnableClientState

2.32 rcIsMaterial

RCboolean rcIsMaterial(RCuint *material*);	
Returns if a material object that has been created with the name, *material*, is present	
material	The name of the material object

Related Functions
rcBindMatreial, rcDeleteMaterials, rcGenMaterials

Explanation

Returns **RC_TRUE** if *material* is currently the name of the material object that has been created. Ortherwise, returns **RC_FALSE** when *material* is non-zero.

2.33 rcIsTexture

RCboolean rcIsTexture(RCuint *texture*);
Returns if a texture object that has been created with the name, *texture*, is present

texture	The name of the texture object

Related Functions
rcBindTexture, rcDeleteTextures, rcGenTextures, rcTexImage2D, rcTexParameter

Explanation

Returns **RC_TRUE** if *texture* is currently the name of the texture object that has been created. Ortherwise, returns **RC_FALSE** when *texture* is non-zero.

2.34 rcLight

void rcLightf(RCenum *light*, RCenum *pname*, RCfloat *param*);
void rcLightx(RCenum *light*, RCenum *pname*, RCfixed *param*);

Sets the information of a light source

light	The light source to which the information is set, RC_LIGHT*i* (0<= *i* < RC_MAX_LIGHTS) (The maximum number of supported light source is 8.)
pname	The information parameter of the light source (RC_SPOT_EXPONENT, RC_SPOT_INNER_CONE, RC_SPOT_OUTER_CONE, RC_SPOT_CUTOFF, RC_START_ATTENUATION, RC_END_ATTENUATION, RC_CONSTANT_ATTENUATION, RC_LINEAR_ATTENUATION, RC_QUADRATIC_ATTENUATION)
param	The value which is set to the information parameter of the light source

void rcLightfv(RCenum *light*, RCenum *pname*, const RCfloat * *params*);
void rcLightxv(RCenum *light*, RCenum *pname*, const RCfixed * *params*);

Sets the information of a light source

light	The light source to which the information is set, RC_LIGHT*i* (0<= *i* < RC_MAX_LIGHTS) (The maximum number of supported light source is 8.)
pname	The information parameter of the light source (RC_AMBIENT, RC_DIFFUSE, RC_SPECULAR, RC_POSITION, RC_SPOT_DIRECTION, RC_SPOT_EXPONENT, RC_SPOT_INNER_CONE, RC_SPOT_OUTER_CONE, RC_SPOT_CUTOFF, RC_START_ATTENUATION, RC_END_ATTENUATION, RC_CONSTANT_ATTENUATION, RC_LINEAR_ATTENUATION, RC_QUADRATIC_ATTENUATION)
params	The pointer to an array of data which is set to the information parameter of the light source

Error Codes

RC_INVALID_ENUM : *light* or *pname* is an invalid value
RC_INVALID_VALUE
: The spot exponent value is negative.
: The spot cutoff is outside the range [0, 90] (except for the special value 180)
: The attenuation coefficient is nagetive

Related Functions

rcEnable, rcMaterial

Attention

RC_LIGHT*i* and RC_LIGHT0+*i* are the same. The alpha value is used in the lighting equation of

RayCore®.

Explanation

Light sources that are enabled contribute to the lighting calculation only when the lighting is enabled. (See **rcEnable** and **rcDisable**) The following is the description of the information parameter of the light source *pname*.

- RC_AMBIENT
  Sets the the ambient RGBA intensity of the light source. The initial ambient light intensity is (1, 1, 1, 0). Specified values are clamped to the range [0, 1].

- RC_DIFFUSE
  Sets the the diffuse RGBA intensity of the light source. The initial value for **RC_LIGHT0** is (0.1, 0.1, 0.1, 0). For other lights, the initial value is (0.1, 0.1, 0.1, 0). Specified values are clamped to the range [0, 1].

- RC_SPECULAR
  Sets the the specular RGBA intensity of the light source. The initial value for **RC_LIGHT0** is (0, 0, 0, 0). For other lights, the initial value is (0, 0, 0, 0). Specified values are clamped to the range [0, 1].

- RC_POSITION
  Sets the position and type of the light source. The position, first 3 values, is transformed by the modelview matrix, and it is stored in the eye coordinate system. If the type *w*, last fourth value, of the light source is 0, the light source is treated as a directional light source. If the type *w* is 1, the light source is treated as a point light source. If the light source is based on the direction vector and the actual position in the eye coordinate system, the light attenuation can be used. The initial position and type is (0, 0, 1, 0); thus, the initial light source is directional, parallel to, and in the direction of the - *z* axis.

- RC_SPOT_DIRECTION
  Sets the direction vector of the light source in the object coordinate system. The direction vector is transformed by the modelview matrix, and it is stored in the eye coordinate system. It is significant only when **RC_SPOT_CUTOFF** is not 180, which it is initially. The initial direction vector is (0, 0, 0).

- RC_SPOT_EXPONENT
  Sets the spot exponent of the light source that is related the intensity distribution. Let Θ be the angle between the direction vector of the light source and the direction from the light to the vertex being lighted. The intensity distribution of the light source is attenuated by the cosine of the angle Θ, raised to the power of the spot exponent. Thus, higher spot exponents result in a more focused light source. The initial spot exponent is 0, resulting in uniform light distribution.

- RC_SPOT_INNER_CONE

Sets the attenuated start angle of the light source. If the angle between the direction of the light and the direction from the light to the vertex being lighted is greater than the spot inner cone angle, the attenuation of light begins. This attenuation intensity is determined by the spot exponent and the attenuation coefficient. Only values in the range [0, 90] and the special value 180 are accepted. The initial value is 0.

- RC_SPOT_OUTER_CONE, RC_SPOT_CUTOFF
  Sets the maximum spread angle of the light source. If the angle between the direction of the light and the direction from the light to the vertex being lighted is less than the spot cutoff angle, the light is continuously attenuated; otherwise, the light is completely masked. This attenuation intensity is determined by the spot exponent and the attenuation coefficient. Only values in the range [0, 90] and the special value 180 are accepted. The initial value is 180, resulting in uniform light distribution.

- RC_ATTENUATION_RANGE
  Sets the distance range from the light position that is attenuated. First value is the start distance, and second value is the end distance.

- RC_START_ATTENUATION
  Sets the distance from the light source to begin the intensity attenuation. The initial value is 0.

- RC_END_ATTENUATION
  Sets the distance from the light source to end the intensity attenuation. The initial value is 0.

- RC_CONSTANT_ATTENUATION, RC_LINEAR_ATTENUATION, RC_QUADRATIC_ATTENUATION
  Sets one of the three attenuation coefficient on the intensity of the light source. These values must be nonnegative and are related the position of the light source. Let *r* be the distance between the light source and the vertex. The intensity of the light source is attenuated by the reciprocal of the sum of the constant factor, the linear factor times *r* and the quadratic factor times *r*. The initial attenuation factors are (1, 0, 0), thus the intensity of the light source according to the distance is uniform.

2.35 rcLoadIdentity

void rcLoadIdentity(void);

Replaces the matrix of the current matrix mode with the 4x4 identity matrix

Related Functions

rcLoadMatrix, rcMatrixMode, rcMultMatrix, rcPushMatrix
rcGet
- RC_MATRIX_MODE
- RC_MODELVIEW_MATRIX
- RC_PROJECTION_MATRIX
- RC_TEXTURE_MATRIX
- RC_MATRIX_PALETTE_OES

Explanation

It is semantically equivalent to calling **rcLoadMatrix** with the 4x4 identity matrix.

The matrix of the current matrix mode is one of the projection matrix, modelview matrix, texture matrix and palette matrix. (See **rcMatrixMode**)

2.36 rcLoadMatrix

void rcLoadMatrixf(const RCfloat * *m*);
void rcLoadMatrixx(const RCfixed * *m*);

Replaces the matrix of the current matrix mode with the specific matrix to which *m* is set

m	The pointer of an array with 16 values which is the elements of a 4x4 column-major matrix

Related Functions

rcLoadIdentity, rcMatrixMode, rcMultMatrix, rcPushMatrix
rcGet
- RC_MATRIX_MODE
- RC_MODELVIEW_MATRIX
- RC_PROJECTION_MATRIX
- RC_TEXTURE_MATRIX
- RC_MATRIX_PALETTE_OES

Explanation

The matrix *M* of the current matrix mode is replaced with the specified array *m* as follows.

$$M=\begin{bmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{bmatrix}$$

The matrix of the current matrix mode is one of the projection matrix, modelview matrix, texture matrix and palette matrix. (See **rcMatrixMode**)

2.37 rcMaterial

void rcMaterialf(RCenum *face*, RCenum *pname*, RCfloat *param*);
void rcMaterialx(RCenum *face*, RCenum *pname*, RCfixed *param*);

Sets the information of the current material object

face	The face of the current material object to which the information is set (RC_FRONT_AND_BACK)
pname	The information parameter of the material object (RC_REFLECTION, RC_REFRATION_INDEX, RC_SHININESS, RC_SPECULAR_LEVEL, RC_TRANSMITTANCE)
param	The value which is set to the information parameter of the material object

void rcMaterialfv(RCenum *face*, RCenum *pname*, const RCfloat * *params*);
void rcMaterialxv(RCenum *face*, RCenum *pname*, const RCfixed * *params*);

Sets the information of the current material object

face	The face of the current material object to which the information is set (RC_FRONT_AND_BACK)
pname	The information parameter of the material object (RC_AMBIENT, RC_AMBIENT_AND_DIFFUSE, RC_DIFFUSE, RC_REFLECTION, RC_REFRATION_INDEX, RC_SHININESS, RC_SPECULAR, RC_SPECULAR_LEVEL, RC_TRANSMITTANCE)
params	The pointer to an array of data which is set to the information parameter of the material object

Error Codes

RC_INVALID_ENUM : *face* or *pname* is an invalid value
RC_INVALID_VALUE
: the specular exponent value is negative
: the specular intensity value is negative

Related Functions

rcEnable, rcGetMaterial, rcLight

Attention

The alpha value is used in the lighting equation of RayCore®.

Explanation

The information of a material object is used in the lighting equation that is applied to the related primitive. The following is the description of the information parameter of the material object *pname*.

- RC_AMBIENT
  Sets the ambient RGBA reflectance of the material object. The initial ambient reflectance is (0.2, 0.2, 0.2, 0). Specified values are clamped to the range [0, 1].

- RC_AMBIENT_AND_DIFFUSE
  Equivalent to calling **rcMaterial** with the same parameter values for each **RC_AMBIENT** and **RC_DIFFUSE**.

- RC_DIFFUSE
  Sets the diffuse RGBA reflectance of the material object. The initial diffuse reflectance is (0.8, 0.8, 0.8, 0). Specified values are clamped to the range [0, 1].

- RC_REFLECTION
  Sets the reflectance of the material object. The initial reflectance is 0.

- RC_REFRACTION_INDEX
  Sets the refraction index of the material object. The initial refraction index is 1.

- RC_SHININESS
  Sets the specular exponent of the material object. The initial specular exponent is 0.

- RC_SPECULAR
  Sets the specular RGBA reflectance of the material object. The initial specular reflectance is (0, 0, 0, 0). Specified values are clamped to the range [0, 1].

- RC_SPECULAR_LEVEL
  Sets the specular intensity of the material object. The initial specular intensity is 0.

- RC_TRANSMITTANCE
  Sets the transmittance of the material object. The initial transmittance is 0.

2.38 rcMatrixMode

void rcMatrixMode(RCenum *mode*);	
Sets the current matrix mode for a matrix to be the current matrix	
mode	The matrix mode to which the current matrix is set (RC_MODELVIEW, RC_PROJECTION, RC_TEXTURE)

Error Codes
RC_INVALID_ENUM : *mode* is an invalid value
Related Functions

rcLoadMatrix, rcMultMatrix, rcPushMatrix
rcGet
RC_MATRIX_MODE

Explanation

If the current matrix mode is set, the current matrix and matrix stack are determined and used in subsequent matrix operations. The initial matrix mode is **RC_MODELVIEW**. The following is the description of a matrix mode *mode*.

- RC_MODELVIEW
  Applies subsequent matrix operations to the current model view matrix and matrix stack.
- RC_PROJECTION
  Applies subsequent matrix operations to the current projection matrix and matrix stack.
- RC_TEXTURE
  Applies subsequent matrix operations to the current texture matrix and matrix stack.
- RC_MATRIX_PALETTE_OES
  Applies subsequent matrix operations to the current palette matrix and matrix stack.

2.39 rcMultMatrix

void rcMultMatrixf(const RCfloat * *m*);
void rcMultMatrixx(const RCfixed * *m*);

Multiplies the matrix of the current matrix mode with the specific matrix to which *m* is set

m	The pointer of an array with 16 values which is the elements of a 4x4 column-major matrix

Related Functions

rcLoadIdentity, rcLoadMatrix, rcMatrixMode, rcPushMatrix
rcGet
- RC_MATRIX_MODE
- RC_MODELVIEW_MATRIX
- RC_PROJECTION_MATRIX
- RC_TEXTURE_MATRIX
- RC_MATRIX_PALETTE_OES

Explanation

The matrix of the current matrix mode is one of the projection matrix, modelview matrix and texture matrix. (See **rcMatrixMode**) By multiplying the matrix M of the current matrix mode with the matrix which is composed of the array m, the new matrix M' of the current matrix mode is created.

$$M' = \begin{bmatrix} M[0] & M[4] & M[8] & M[12] \\ M[1] & M[5] & M[9] & M[13] \\ M[2] & M[6] & M[10] & M[14] \\ M[3] & M[7] & M[11] & M[15] \end{bmatrix} \begin{bmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{bmatrix}$$

2.40 rcNormalPointer

Void rcNormalPointer(RCenum *type*, RCsizei *stride*, const RCvoid * *pointer*);	
Sets the information of a normals array to be used in rendering	
type	The data type of a normals array (RC_BYTE, RC_SHORT, RC_FLOAT, RC_FIXED)
stride	The byte length between two adjacent normals (If this value is 0, the normals are tightly arranged in the array.)
pointer	The data pointer of the first component for the first normal in the array

Error Codes
RC_INVALID_ENUM : *type* is an invalid value **RC_INVALID_VALUE : *stride* is negative**
Related Functions
rcDrawArrays, rcDrawElements, rcEnable, rcTexCoordPointer, rcVertexPointer

Explanation

The information of normals can be used after being packed into a single array or stored in separate arrays along with colors, vertices, normals, and texture coordinates. If enabled, the normals array is used when **rcDrawArrays** or **rcDrawElements** is called. (See **rcEnableClientState** and **rcDisableClientState**)

2.41 rcPushMatrix, rcPopMatrix

void rcPushMatrix(void);
void rcPopMatrix(void);

Pushes and pops the matrix stack of the current matrix mode

Error Codes
RC_STACK_OVERFLOW : The current matrix stack is full when rcPushMatrix is called **RC_STACK_UNDERFLOW : The current matrix stack is empty when rcPopMatrix is called**

Related Functions
rcLoadIdentity, rcLoadMatrix, rcMatrixMode, rcMultMatrix, rcRotate, rcScale, rcTranslate rcGet RC_MAX_MODELVIEW_STACK_DEPTH RC_MAX_PROJECTION_STACK_DEPTH RC_MAX_TEXTURE_STACK_DEPTH RC_MAX_TEXTURE_UNITS

Explanation

rcPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. **rcPopMatrix** pops the current matrix stack, replacing the current matrix with the the matrix on top of the current matrix stack. Initially, each of the stacks is empty.

2.42 rcRotate

void rcRotatef(RCfloat *angle*, RCfloat *x*, RCfloat *y*, RCfloat *z*);
void rcRotatex(RCfixed *angle*, RCfixed *x*, RCfixed *y*, RCfixed *z*);

Multiplies the matrix of the current matrix mode with a rotation matrix

angle	The angle of rotation, in degrees
x, y, z	The *x*, *y*, and *z* coordinates of a vector that is the basis of rotation

Related Functions

rcMatrixMode, rcMultMatrix, rcPushMatrix, rcScale, rcTranslate
rcGet
RC_MATRIX_MODE
RC_MODELVIEW_MATRIX
RC_PROJECTION_MATRIX
RC_TEXTURE_MATRIX

Attention

This rotation follows the right-hand rule in RayCore® API, so if the vector (*x, y, z*) points toward the user, the direction of rotation is counterclock wise.

Explanation

The matrix of the current matrix mode is one of the projection matrix, modelview matrix and texture matrix. (See **rcMatrixMode**) The defined rotation matrix *R* is the angle of rotation in degrees, *angle*, around the vector (*x, y, z*). The size of the vector (*x, y, z*) is normalized to 1. Let *a* be cos(*angle*), and *b* be sin(*angle*). The rotation matrix *R* can be expressed as follows:

$$R=\begin{bmatrix} (1-a)x^2+a & (1-a)xy-bz & (1-a)xz+by & 0 \\ (1-a)xy+bz & (1-a)y^2+ay & (1-a)yz-bx & 0 \\ (1-a)xz-by & (1-a)yz+bx & (1-a)z^2+a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rcPushMatrix and **rcPopMatrix** can be used to save and restore the unrotated coordinate system when needed.

2.43 rcScale

void rcScalef(RCfloat *x*, RCfloat *y*, RCfloat *z*);
void rcScalex(RCfixed *x*, RCfixed *y*, RCfixed *z*);

Multiplies the matrix of the current matrix mode with a scaling matrix

x*, *y*, *z	The scale coefficients along the *x*, *y* and *z* axes

Related Functions

rcEnable, rcMatrixMode, rcMultMatrix, rcPushMatrix, rcRotate, rcTranslate
rcGet
RC_MATRIX_MODE
RC_MODELVIEW_MATRIX
RC_PROJECTION_MATRIX
RC_TEXTURE_MATRIX

Attention

If the scaling matrix is applied to the modelview matrix, the information of the light source often appears wrong.

Explanation

The matrix of the current matrix mode is one of the projection matrix, modelview matrix and texture matrix. (See **rcMatrixMode**) The defined scaling matrix produces a nonuniform scaling along the *x*, *y* and *z* axes. The scaling matrix *S* can be expressed as follows:

$$S=\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rcPushMatrix and rcPopMatrix can be used to save and restore the unscaled coordinate system when needed.

2.44 rcTexCoordPointer

void rcTexCoordPointer(RCint *size*, RCenum *type*, RCsizei *stride*, const RCvoid * *pointer*);	
Set the information of a texture coordinates array to be used in rendering	
size	The count of components per texture coordinate (Only 2)
type	The data type of a texture coordinates array (RC_BYTE, RC_SHORT, RC_FLOAT, RC_FIXED)
stride	The byte length between two adjacent texture coordinates (If this value is 0, the texture coordinates are tightly arranged in the array.)
pointer	The data pointer of the first component for the first texture coordinate in the array

Error Codes
RC_INVALID_VALUE *: size* is not 2 **RC_INVALID_ENUM : *type* is an invalid value** **RC_INVALID_VALUE : *stride* is negative**
Related Functions
rcDrawArrays, rcDrawElements, rcEnable, rcNormalPointer, rcVertexPointer

Explanation

The information of texture coordinates may be stored in a single array or separate arrys, along with colors, vertices, normals, and texture coordinates. If enabled, the texture coordinates array is used when **rcDrawArrays** or **rcDrawElements** is called. (See rcEnableClientState and rcDisableClientState)

2.45 rcTexImage2D

void rcTexImage2D(RCenum *target*, RCint *level*, RCint *internalformat*, RCsizei *width*, RCsizei *height*, RCint *border*, RCenum *format*, RCenum *type*, const RCvoid * *pixels*);

Sets the two-dimensional texture image to the current texture object

target	The texture target (RC_TEXTURE_2D, RC_TEXTURE_2D_NORMAL)
level	The level number of LOD(level-of-detail) (Only 0)
internalformat	The type of the color components in the texture (RC_RGB, RC_RGBA)
width	The width of the texture image (Must be a multiple of 2, At least a minimum of 16, and Up to RC_MAX_TEXTURE_SIZE or less)
height	The height of the texture image (Must be a multiple of 2, At least a minimum of 16, and Up to RC_MAX_TEXTURE_SIZE or less)
border	The width of the border (Only 0)
format	The type of the color components in the pixel data (RC_RGB, RC_RGBA)
type	The data type of the pixel data (RC_BYTE, RC_UNSIGNED_BYTE, RC_SHORT, RC_UNSIGNED_SHORT)
pixels	The memory address where the pixel data is loaded

Error Codes

RC_INVALID_ENUM
: *target* is an invalid value
: *internalformat* or *format* is an invalid value : *type* is an invalid value
RC_INVALID_VALUE
: *level* is not zero

: *width* or *height* is less than 16 or greater than RC_MAX_TEXTURE_SIZE, or is not a multiple of 2
: *border* is not zero
RC_INVALID_OPERATION *: internalformat* and *format* are not the same

Related Functions

rcBindTexture,
rcGet
RC_MAX_TEXTURE_SIZE

Attention

If *pixels* is **NULL**, texturing cannot be applied.

Explanation

If the type of the color components is **RC_RGB**, it is converted and assembled into an RGBA element by attaching an alpha value which is specified by calling **rcTextureAlpha** in RayCore® API. The types of the color components, *internalformat* and *format,* are as follows:

- RC_RGB
  Each color is composed of an RGB (the red, green and blue values).

- RC_RGBA
  Each color is composed of an RGBA (the red, green, blue and alpha values).

2.46 rcTexParameter

void rcTexParameterf(RCenum *target*, RCenum *pname*, RCfloat *param*); void rcTexParameteri(RCenum *target*, RCenum *pname*, RCint *param*); void rcTexParameterx(RCenum *target*, RCenum *pname*, RCfixed *param*);	
Sets the information of a texture target	
target	The texture target to which information is set (RC_TEXTURE_2D, RC_TEXTURE_2D_NORMAL)
pname	The texture target to which information is set (RC_TEXTURE_WRAP_S, RC_TEXTURE_WRAP_T, RC_GENERATE_MIPMAP)
param	The value which is set to the information parameter of the texture target

void rcTexParameterfv(RCenum target, RCenum pname, RCfloat * params); void rcTexParameteriv(RCenum target, RCenum pname, RCint * params); void rcTexParameterxv(RCenum target, RCenum pname, RCfixed * params);	
Sets the information of a texture target	
target	The texture target to which information is set (RC_TEXTURE_2D, RC_TEXTURE_2D_NORMAL)
pname	The information parameter of the texture target (RC_TEXTURE_WRAP_S, RC_TEXTURE_WRAP_T, RC_GENERATE_MIPMAP)
params	The pointer to an array of data which is set to the information parameter of the texture target

Error Codes
RC_INVALID_ENUM *: target* or *pname* is an invalid value
Related Functions
rcBindTexture, rcEnable, rcTexImage2D, rcTexSubImage2D

Explanation

The following is the description of the information parameter of the texture target *pname*.

- RC_TEXTURE_WRAP_S
  Sets the wrap parameter for texture coordinate *s* to either **RC_CLAMP_TO_EDGE** or **RC_REPEAT**. The initial value is **RC_REPEAT**.

- RC_CLAMP_TO_EDGE
  It causes *s* coordinates to be clamped to the range [1/2*N*, 1 – 1/2*N*], where *N* is the size of the texture in the direction of clamping.

- RC_REPEAT

  It causes the integer part of the *s* coordinate to be ignored; only the fractional part is used, thereby creating a repeating pattern.

- RC_TEXTURE_WRAP_T

  Sets the wrap parameter for texture coordinate *t* to either **RC_CLAMP_TO_EDGE** or **RC_REPEAT**. See the discussion under **RC_TEXTURE_WRAP_S**. The initial value is **RC_REPEAT**.

- RC_GENERATE_MIPMAP

  Sets the automatic mipmap generation parameter. If set to **RC_TRUE**, all levels of a mipmap array should be automatically updated when any modification to the base level mipmap is done. The initial value is **RC_TRUE**.

2.47 rcTranslate

void rcTranslatef(RCfloat *x*, RCfloat *y*, RCfloat *z*); **void rcTranslatex(RCfixed *x*, RCfixed *y*, RCfixed *z*);**	
Multiplies the matrix of the current matrix mode with a translation matrix	
x, y, z	The *x*, *y*, and *z* coordinates of a translation vector

Related Functions

rcMatrixMode, rcMultMatrix, rcPushMatrix, rcRotate, rcScale
rcGet
- RC_MATRIX_MODE
- RC_MODELVIEW_MATRIX
- RC_PROJECTION_MATRIX
- RC_TEXTURE_MATRIX

Explanation

The matrix of the current matrix mode is one of the projection matrix, modelview matrix and texture matrix. (See **rcMatrixMode**) The defined translation matrix *T* produces a translation by vector (*x, y, z*). The translation matrix *T* can be expressed as follows:

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rcPushMatrix and **rcPopMatrix** can be used to save and store the untranslated coordinate system when needed.

2.48 rcVertexPointer

void rcVertexPointer(RCint *size*, RCenum *type*, RCsizei *stride*, const RCvoid * *pointer*);	
Sets the information of a vertices array to be used in rendering	
size	The count of coordinates per vertex (Only 3)
type	The data type of a vertices array (RC_BYTE, RC_SHORT, RC_FLOAT, RC_FIXED)
stride	The byte length between two adjacent vertices (If this value is 0, the vertices are tightly arranged in the array.)
pointer	The data pointer of the first coordinate for the first vertex in the array

Error Codes
RC_INVALID_ENUM : *type* is an invalid value RC_INVALID_VALUE **: *size* is not 3** **: *stride* is negative**
Related Functions
rcDrawArrays, rcDrawElements, rcEnable, rcNormalPointer, rcTexCoordPointer

Explanation

The information of vertex coordinates may be stored in a single array or separate arrays, along with colors, vertices, normals, and texture coordinates.

If enabled, the vertex coordinates array is used when **rcDrawArrays** or **rcDrawElements** is called. (See **rcEnableClientState** and **rcDisableClientState**)

2.49 rcViewport

void rcViewport(RCint *x*, RCint *y*, RCsizei *width*, RCsizei *height*);	
Sets the screen information of a viewport	
x, y	The pixel coordinates of the lower left corner in the viewport rectangle (Only (0, 0))
width, height	The width and height of the viewport rectangle in pixels

Error Codes
RC_INVALID_VALUE : *width* or *height* is negative
Related Functions
rcGet

Explanation

width and *height* are screen resolution, from which the number of ray to be generated is determined. By calling **rcGet** with argument **RC_MAX_VIEWPORT_DIMS**, it querys the range to which the width and height of the viewport are clamped.

Chapter 3
Extended Function List

This chapter covers the extended function list of RayCore® API. Most of the functions are similarly defined as those of OpenGL ES 1.1 with some of them added or modified for RayCore®.

3.1 rcuLookAt

void rcuLookAt(RCfloat *eyex*, RCfloat *eyey*, RCfloat *eyez*, RCfloat *centerx*, RCfloat *centery*, RCfloat *centerz*, RCfloat *upx*, RCfloat *upy*, RCfloat *upz*);	
Sets the camera coordinate system with viewing information	
eyeX, eyeY, eyeZ	The position of the eye point
centerX, centerY, centerZ	The position of the reference point, indicating the center of the scene
upX, upY, upZ	The direction of the *UP* vector

Related Functions

rcFrustum , rcuPerspective

Explanation

The camera coordinate system is defined from an eye point, a reference point, and an *UP* vector. In the camera coordinate system, *X*-axis is called *RIGHT* vector, *Y*-axis is called *UP* vector, and *Z*-axis is called *GAZE* vector.

Let

$$G=\begin{bmatrix} eyeX-centerX \\ eyeY-centerY \\ eyeZ-centerZ \end{bmatrix}$$

Let *UP* be the vector (*upX*, *upY*, *upZ*).

Then normalize as follows:

$$g=\frac{G}{¿|G|\vee ¿¿}$$

$$u=\frac{UP}{¿|UP|\vee ¿¿}$$

Finally, let *R* be the cross product of *u* and *f*.

$$R=u\times f$$

$$r=\frac{R}{¿|R|\vee ¿¿}$$

These represent the reference axes of a camera coordinate system. *r* is the unit vector for *X*-axis, *u* is

the unit vector for *Y*-axis, and *f* is the unit vector for *Z*-axis.

3.2 rcuPerspective

void rcuPerspective(RCfloat *fovy*, RCfloat *aspect*, RCfloat *zNear*, RCfloat *zFar*);	
Sets the viewing frustum into the world coordinate system	
fovy	The field of view angle, in degrees, in the y direction
aspect	The aspect ratio that determines the field of view in the *x* direction (The ratio of *x* (width) to *y* (height))
zNear	The distance from the viewer to the near clipping plane (always positive)
zFar	The distance from the viewer to the far clipping plane (always positive)

Error Codes
RC_INVALID_VALUE **: *zNear* or *zFar* is not positive** **: *zNear* is equal to *zFar***
Related Functions
rcFrustum, rcLoadIdentity, rcMultMatrix

Attention

zNear must never be 0.

Explanation

The aspect ratio should match the aspect ratio of the viewport. *aspect*=2.0 means that the viewer's angle of view is twice as wide in *x* as it is in *y*. If the viewport is twice as wide as it is tall, it displays the image without distortion.

Given *f* is defined as follows:

$$f = tangent\left(\frac{fovy}{2}\right)$$

Let the width of a screen be *Xmax*, and the height of a screen be *Ymax.*

$$Ymax = 2 \times zNear \times f$$

$$Xmax = Ymax \times aspect$$

The size in pixels of a clipping plane which represents the screen is caculated with these values. Here, *zNear* is a distance from the camera to the screen. *zFar* is not used.

3.3 rcCurrentPaletteMatrixOES

void rcCurrentPaletteMatrixOES(RCuint *index*);	
Sets the current palette matrix that is used on subsequent matrix operations	
index	The index of palette matrices

Error Codes
RC_INVALID_VALUE : *index* is greater than RC_MAX_PALETTE_MATRICES_OES – 1
Related Functions
rcLoadPaletteFromModelViewMatrixOES, rcMatrixIndexPointerOES, rcMatrixMode, rcWeightPointerOES

Explanation

rcCurrentPaletteMatrixOES can be used when the current matrix mode is **RC_MATRIX_PALETTE_OES**.

3.4 rcLoadPaletteFromModelViewMatrixOES

void rcLoadPaletteFromModelViewMatrixOES(void);
Copies the current model view matrix to the current palette matrix

Related Functions
rcCurrentPaletteMatrixOES, rcMatrixIndexPointerOES, rcMatrixMode, rcWeightPointerOES

Explanation

The current palette matrix is assigned by **rcCurrentPaletteMatrixOES**.

3.5 rcMatrixIndexPointerOES

void rcMatrixIndexPointerOES(RCint *size*, RCenum *type*, RCsizei *stride*, const RCvoid **pointer*);

Sets the information of a matrix indices array to be used in rendering

size	The count of matrix indices per vertex (The maximum count is RC_MAX_VERTEX_UNITS_OES)
type	The data type of a matrix indices array (RC_UNSIGNED_BYTE, RC_UNSIGNED_INT)
stride	The byte length between two adjacent matrix indices (If this value is 0, the matrix indices are tightly arranged in the array.)
pointer	The data pointer of the first matrix index for the first vertex in the array

Error Codes

RC_INVALID_ENUM : *type* is an invalid value

RC_INVALID_VALUE

: *size* is 0, negative, or greater than RC_MAX_VERTEX_UNITS_OES

: *stride* is negative

Related Functions

rcCurrentPaletteMatrixOES, rcDrawArrays, rcDrawElements, rcLoadPaletteFromModelViewMatrixOES, rcMatrixMode, rcWeightPointerOES

Explanation

These matrix indices are used to mix corresponding matrices for a given vertex. The enabled matrix indices array is used when **rcDrawArrays** or **rcDrawElements** is called. (See **rcEnableClientState** and **rcDisableClientState**)

3.6 rcWeightPointerOES

void rcWeightPointerOES(RCint *size*, RCenum *type*, RCsizei *stride*, const RCvoid **pointer*);

Sets the information of a weights array to be used in rendering

size	The count of weights per vertex (The maximum count is RC_MAX_VERTEX_UNITS_OES)
type	The data type of a weights array (RC_FIXED, RC_FLOAT)
stride	The byte length between two adjacent weights (If this value is 0, the weights are tightly arranged in the array.)
pointer	The data pointer of the first weight for the first vertex in the array

Error Codes

RC_INVALID_ENUM : *type* is an invalid value

RC_INVALID_VALUE

: *size* is 0, negative, or greater than RC_MAX_VERTEX_UNITS_OES

: *stride* is negative

Related Functions

rcCurrentPaletteMatrixOES, rcDrawArrays, rcDrawElements, rcLoadPaletteFromModelViewMatrixOES, rcMatrixIndexPointerOES, rcMatrixMode

Explanation

These weights are used to mix corresponding matrices for a given vertex. The enabled weights array is used when **rcDrawArrays**, or **rcDrawElements** is called. (See **rcEnableClientState** and **rcDisableClientState**)

3.7 rcSceneAllInit

void rcSceneAllInit(void);
Initializes both static and dynamic scene data

Related Functions
rcStaticSceneBegin, rcStaticSceneEnd, rcFinish

Explanation

rcSceneAllInit initializes the rendering data of all primitives in the static scene, which is delimited in between **rcStaticSceneBegin** and **rcStaticSceneEnd**.

To redraw all the scenes, call **rcSceneAllInit**.

3.8 rcStaticSceneBegin, rcStaticSceneEnd

void rcStaticSceneBegin(void);

void rcStaticSceneEnd(void);

Delimits the vertices, material and texture of the primitive or a group of the primitives in the static scene

Related Functions

rcVertexPointer, rcTexCoordPointer, rcNormalPointer, rcDrawArrays, rcDrswElements, rcBindMaterial, rcMaterial, rcBindTexture, rcFinish

Attention

rcStaticScenenEnd must be called after **rcStaticSceneBegin** is called. Once the primitives in the static scene are delimited, they are rendered continuously until **rcSceneAllInit** or **rcStaticSceneBegin** is called again.

Explanation

rcStaticSceneBegin and **rcStaticSceneEnd** delimit the vertices of the primitive or a group of the primitives.

Only a subset of RC commands can be used between rcStaticSceneBegin and rcStaticSceneEnd. The commands are rcVertexPointer, rcTexCoordPointer, rcNormalPointer, rcDrawArrays, rcDrswElements, rcBindMaterial, rcMaterial and rcBindTexture.

There is no limit to the number of vertices that can be defined between **rcStaticSceneBegin** and **rcStaticSceneEnd**. Triangles and quadrilaterals that are incompletely specified are not drawn.

The minimum specification of vertices for each primitive is as follows:

- 3 for a triangle and 4 for a quadrilateral.
- Modes that require a certain multiple of vertices are RC_TRIANGLES(3), RC_TRIANGLE_STRIP(3), RC_TRIANGLE_FAN(3), and RC_QUADS(4).

3.9 rcTextureAlpha

void rcTextureAlpha(RCbyte *value*);	
Specifies an alpha value for the alpha channel of texture with RGB data format	
value	The additional alpha value used when the material texture is without its alpha value (Initial value : 16)

Related Functions
rcBindMaterial, rcMaterial, rcBindTexture, rcTexImage2D

Explanation

rcTextureAlpha specifies the alpha value used by **rcTexImage2D** to set the additional alpha value of RGB format texture without the alpha value. Values specified by **rcTextureAlpha** are clamped to the range [0, 255].

3.10 rcDepthBounce

void rcDepthBounce(RCuint *value*);	
Sets the depth of the ray bounce	
value	The depth of the ray bounce used when the ray is generated in ray tracing. (Initial value : 10)

Related Functions
rcFinish

Explanation

rcDepthBounce specifies the maximum bounce depth of the ray generated by the ray tracing rendering process. Values specified by **rcDepthBounce** are clamped to the range [0, 14].