CRay Engine Mini

Generated by Doxygen 1.8.16

Chapter 1

Main Page

1.1 Introduction

This is a basic CPU ray tracing engine, consisting of two basic files and with a minimum of dependencies. The main idea was to be able to quickly render an image through simple API calls for rapid prototyping, that could be easily integrated and exploited through a larger engine. Since other, usually heavy utilities, responsible for image loading, logging, geometry I/O, etc. are either already present as part of larger tools or exist as separate projects, the provided engine is responsible only for the pure rendering part, i.e. it is agnostic on how images and external models have been loaded.

1.2 Features

List of features:

- C/C++ API
- Multithreaded tiled rendering
- BVH for storing the scene primitives
- Two integrators (for now): ambient occlusion and path tracing
- Cameras: perspective, thinlens and orthographic
- Primitives: spheres, rectangles, triangles
- Multiple materials for modelling the typical matte, glass, plastic surfaces, etc (see the list of supported materials below). In more detail:
  - Support for diffuse/specular components and smooth/rough reflection/transmission
  - Diffuse surfaces are modelled as pure Lambertian and specular using isotropic GGX.
- Light sources:
  - diffuse area lights
  - point spot/omnilights
  - directional lights
- Multiple importance sampling
- Basic texturing: box and bilinear filtering
- Support for hierarchical transformations
- Interactive (progressive) rendering, to be integrated in a GUI for navigation
- Internal logger mechanism as well as rendering progress functionality

1.3 Credits

Kostas Vardis
Website: https://kostasvardis.com
Any acknowledgments/references are appreciated.

1.4 License

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Base Objects and Handles

Representations for base engine objects.

Classes

- struct cr_vec3

  Struct for a 3-dimensional vector.
- struct cr_mat4

  Struct for a 4x4 matrix.

Macros

- #define DOUBLE_PRECISION

  Define to set double precision floats as default.
- #define VEC3(_x, _y, _z) (cr_vec3){_x,_y,_z}

  Vec3 constructor from 3 values.
- #define VEC3_XYZ(_x) (cr_vec3){_x,_x,_x}

  Vec3 constructor from 1 value.
- #define VEC3_ARRAY(_x) (cr_vec3){_x[0],_x[1],_x[2]}

  Vec3 constructor from array.
- #define MAT4(_x0, _y0, _z0, _w0, _x1, _y1, _z1, _w1, _x2, _y2, _z2, _w2, _x3, _y3, _z3, _w3) (cr_mat4){↩ _x0, _y0, _z0, _w0, _x1, _y1, _z1, _w1, _x2, _y2, _z2, _w2, _x3, _y3, _z3, _w3}

  Mat4 constructor from 16 values.
- #define MAT4_IDENTITY (cr_mat4){1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1}

  Mat4 constructor for identity matrix.
- #define CRAY_HANDLE uint64_t

  Opaque handle.
- #define CRAY_INVALID_HANDLE 0u

  Opaque invalid handle.

Typedefs

- typedef double cr_float

  floating point value
- typedef struct cr_vec3 cr_vec3

  Struct for a 3-dimensional vector.
- typedef struct cr_mat4 cr_mat4

  Struct for a 4x4 matrix.

5.1.1 Detailed Description

Representations for base engine objects.

5.1.2 Macro Definition Documentation

5.1.2.1 CRAY_HANDLE

```
#define CRAY_HANDLE uint64_t
```

Opaque handle.

5.1.2.2 CRAY_INVALID_HANDLE

```
#define CRAY_INVALID_HANDLE 0u
```

Opaque invalid handle.

5.1.2.3 DOUBLE_PRECISION

```
#define DOUBLE_PRECISION
```

Define to set double precision floats as default.

5.1.2.4 MAT4

```
#define MAT4(
              _x0,
              _y0,
              _z0,
              _w0,
              _x1,
              _y1,
              _z1,
              _w1,
              _x2,
              _y2,
              _z2,
              _w2,
              _x3,
              _y3,
              _z3,
              _w3 ) (cr_mat4){_x0, _y0, _z0, _w0, _x1, _y1, _z1, _w1, _x2, _y2, _z2, _w2, _x3,
_y3, _z3, _w3}
```

Mat4 constructor from 16 values.

5.1.2.5 MAT4_IDENTITY

```
#define MAT4_IDENTITY (cr_mat4){1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1}
```

Mat4 constructor for identity matrix.

5.1.2.6 VEC3

```
#define VEC3(
              _x,
              _y,
              _z ) (cr_vec3){_x,_y,_z}
```

Vec3 constructor from 3 values.

5.1.2.7 VEC3_ARRAY

```
#define VEC3_ARRAY(
              _x ) (cr_vec3){_x[0],_x[1],_x[2]}
```

Vec3 constructor from array.

5.1.2.8 VEC3_XYZ

```
#define VEC3_XYZ(
              _x ) (cr_vec3){_x,_x,_x}
```

Vec3 constructor from 1 value.

5.1.3 Typedef Documentation

5.1.3.1 cr_float

```
typedef double cr_float
```

floating point value

5.1.3.2 cr_mat4

```
typedef struct cr_mat4 cr_mat4
```

Struct for a 4x4 matrix.

5.1.3.3 cr_vec3

```
typedef struct cr_vec3 cr_vec3
```

Struct for a 3-dimensional vector.

5.2 Cameras API

Functions for the generation of cameras.

Functions

- void CR_addCameraThinLens (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float aperture, cr_float far, cr_float lens_radius, cr_float focal_distance)

  Creates a thinlens camera. This is essentially the perspective camera with a predefined lens radius and focal distance.
- void CR_addCameraPerspective (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float aperture, cr_float near, cr_float far)

  Creates a perspective pinhole camera. This is essentially the thinlens camera without a predefined lens radius and focal distance.
- void CR_addCameraOrthographic (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float height, cr_float near, cr_float far)

  Creates an orthographic camera.

5.2.1 Detailed Description

Functions for the generation of cameras.

5.2.2 Function Documentation

5.2.2.1 CR_addCameraOrthographic()

```
void CR_addCameraOrthographic (
            cr_vec3 position,
            cr_vec3 target,
            cr_vec3 up,
            cr_float height,
            cr_float near,
            cr_float far )
```

Creates an orthographic camera.

Parameters

position	The position of the camera in world units.
target	The target of the camera in world units.
up	The up vector of the camera. This does not need to be accurate as it gets recalculated internally.
height	The height of the viewport, in world units. The width is automatically adjusted based on the aspect ration
near	The near field of the camera. This is not necessary, but is provided as an extra parameter.
far	The far field of the camera. This is not necessary, but is provided as an extra parameter.

5.2.2.2 CR_addCameraPerspective()

```
void CR_addCameraPerspective (
            cr_vec3 position,
            cr_vec3 target,
            cr_vec3 up,
            cr_float aperture,
            cr_float near,
            cr_float far )
```

Creates a perspective pinhole camera. This is essentially the thinlens camera without a predefined lens radius and focal distance.

Parameters

position	The position of the camera in world units.
target	The target of the camera in world units.
up	The up vector of the camera. This does not need to be accurate as it gets recalculated internally.
aperture	The vertical FOV of the camera.
near	The near field of the camera. This is not necessary, but is provided as an extra parameter.
far	The far field of the camera. This is not necessary, but is provided as an extra parameter.

5.2.2.3 CR_addCameraThinLens()

```
void CR_addCameraThinLens (
            cr_vec3 position,
            cr_vec3 target,
            cr_vec3 up,
            cr_float aperture,
            cr_float far,
            cr_float lens_radius,
            cr_float focal_distance )
```

Creates a thinlens camera. This is essentially the perspective camera with a predefined lens radius and focal distance.

Parameters

position	The position of the camera in world units.
target	The target of the camera in world units.
up	The up vector of the camera. This does not need to be accurate as it gets recalculated internally.
aperture	The vertical FOV of the camera.
far	The far field of the camera. This is not necessary, but is provided as an extra parameter.
lens_radius	The spherical radius of the thinlens aperture. Setting this to 0 delegates the thinlens camera to the perspective pinhole camera.
focal_distance	The focal distance of the camera, the plane where the objects are in focus. Setting this to 0 delegates the thinlens camera to the perspective pinhole camera.

5.3 Debug API

Debugging functionality. Used internally for testing purposes.

Functions

- bool CR_debug_intersectPixel (cr_vec3 *position, int32_t pixel_x, int32_t pixel_y)

  Debug utility that returns the intersection position at a particular pixel.
- void CR_debug_updateCameraPerspective (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float aperture, cr_float near, cr_float far)

  Debug utility that updates camera parameters for navigation during the interactive mode. This is only useful as part of a GUI.
- void CR_debug_setTestPixelRange (int32_t start_x, int32_t start_y, int32_t end_x, int32_t end_y)

  Debug utility that logs verbose messages in debug mode for a particular pixel range [start, stop]. Very useful for cases where internal values need to be viewed. Enable/Disable with CR_debug_setTestPixelRange.
- void CR_debug_enableTestPixelDebug (bool enable)

  Debug utility to enable logging of verbose messages in debug mode for a particular pixel range [start, stop].
- bool CR_debug_isTestPixelDebugEnabled ()

  Debug utility that returns if debug pixel printing is enabled.

5.3.1 Detailed Description

Debugging functionality. Used internally for testing purposes.

5.3.2 Function Documentation

5.3.2.1 CR_debug_enableTestPixelDebug()

```
void CR_debug_enableTestPixelDebug (
            bool enable )
```

Debug utility to enable logging of verbose messages in debug mode for a particular pixel range [start, stop].

Parameters

enable	A flag indicating whether debug pixel info will be printed for the given range. (Default: false)

5.3.2.2 CR_debug_intersectPixel()

```
bool CR_debug_intersectPixel (
            cr_vec3 * position,
```

```
            int32_t pixel_x,
            int32_t pixel_y )
```

Debug utility that returns the intersection position at a particular pixel.

Parameters

position	Stores the intersection position.
pixel_x	The x pixel to test for intersection.
pixel_y	The y pixel to test for intersection.

Returns

Returns true if there is an intersection, false otherwise.

5.3.2.3 CR_debug_isTestPixelDebugEnabled()

```
bool CR_debug_isTestPixelDebugEnabled ( )
```

Debug utility that returns if debug pixel printing is enabled.

5.3.2.4 CR_debug_setTestPixelRange()

```
void CR_debug_setTestPixelRange (
            int32_t start_x,
            int32_t start_y,
            int32_t end_x,
            int32_t end_y )
```

Debug utility that logs verbose messages in debug mode for a particular pixel range [start, stop]. Very useful for cases where internal values need to be viewed. Enable/Disable with CR_debug_setTestPixelRange.

Parameters

start_x	The start pixel x value to start logging (Default: none).
start_y	The start pixel y value to start logging (Default: none).
end_x	The end pixel x value to stop logging (Default: none).
end_y	The end pixel y value to stop logging (Default: none).

5.3.2.5 CR_debug_updateCameraPerspective()

```
void CR_debug_updateCameraPerspective (
            cr_vec3 position,
            cr_vec3 target,
            cr_vec3 up,
            cr_float aperture,
            cr_float near,
            cr_float far )
```

Debug utility that updates camera parameters for navigation during the interactive mode. This is only useful as part of a GUI.

Parameters

position	The position of the camera in world units.
target	The target of the camera in world units.
up	The up vector of the camera. This does not need to be accurate as it gets recalculated internally.
aperture	The vertical FOV of the camera.
near	The near field of the camera. This is not necessary, but is provided as an extra parameter.
far	The far field of the camera. This is not necessary, but is provided as an extra parameter.

5.4 Enums

Defines the enums used in the engine to define groups of elements for various operations.

Typedefs

- typedef enum CRAY_ACCELERATION_STRUCTURE CRAY_ACCELERATION_STRUCTURE

  List of supported acceleration structures.
- typedef enum CRAY_INTEGRATOR CRAY_INTEGRATOR

  List of supported integrators.
- typedef enum CRAY_LOGGERENTRY CRAY_LOGGERENTRY

  List of supported log messages.
- typedef enum CRAY_IOR_DIELECTRICS CRAY_IOR_DIELECTRICS

  List of IOR for supported dielectrics, sampled at wavelength 526nm. Source: https://refractiveindex.info.
- typedef enum CRAY_IOR_CONDUCTORS CRAY_IOR_CONDUCTORS

  List of supported conductors (IOR and absorption), sampled at wavelengths 645nm, 526nm, 444nm. Source: https://refractiveindex.info.
- typedef enum CRAY_TEXTURE_WRAPPING CRAY_TEXTURE_WRAPPING

  List of supported texture wrapping modes.
- typedef enum CRAY_TEXTURE_MAPPING CRAY_TEXTURE_MAPPING

  List of supported texture mapping modes.
- typedef enum CRAY_TEXTURE_FILTERING CRAY_TEXTURE_FILTERING

  List of supported texture filtering modes.

Enumerations

- enum CRAY_ACCELERATION_STRUCTURE { CR_AS_ARRAY, CR_AS_BVH, CR_AS_NONE }

  List of supported acceleration structures.
- enum CRAY_INTEGRATOR { CR_INTEGRATOR_PT, CR_INTEGRATOR_AO, CR_INTEGRATOR_NONE }

  List of supported integrators.
- enum CRAY_LOGGERENTRY { CR_LOGGER_ERROR, CR_LOGGER_WARNING, CR_LOGGER_INFO, CR_LOGGER_ASSERT, CR_LOGGER_DEBUG, CR_LOGGER_NOTHING }

  List of supported log messages.
- enum CRAY_IOR_DIELECTRICS { CR_DIELECTRIC_Acrylic_glass, CR_DIELECTRIC_Polystyrene, CR_DIELECTRIC_Polycarbonate, CR_DIELECTRIC_Diamond, CR_DIELECTRIC_Ice, CR_DIELECTRIC_Sapphire, CR_DIELECTRIC_Crown_Glass_bk7, CR_DIELECTRIC_Soda_lime_gla CR_DIELECTRIC_Water_25C, CR_DIELECTRIC_Acetone_20C, CR_DIELECTRIC_Air, CR_DIELECTRIC_Carbon_dioxide, CR_DIELECTRIC_ALL_DIELECTRICS }

  List of IOR for supported dielectrics, sampled at wavelength 526nm. Source: https://refractiveindex.info.
- enum CRAY_IOR_CONDUCTORS { CR_CONDUCTOR_Alluminium_Al, CR_CONDUCTOR_Brass_CuZn, CR_CONDUCTOR_Copper_Cu, CR_CONDUCTOR_Gold_Au, CR_CONDUCTOR_Iron_Fe, CR_CONDUCTOR_Silver_Ag, CR_CONDUCTOR_ALL_CONDUCTORS }

  List of supported conductors (IOR and absorption), sampled at wavelengths 645nm, 526nm, 444nm. Source: https://refractiveindex.info.
- enum CRAY_TEXTURE_WRAPPING { CTW_CLAMP, CTW_REPEAT }

List of supported texture wrapping modes.

- enum CRAY_TEXTURE_MAPPING { CTM_UV, CTM_PLANAR, CTM_SPHERICAL }

  List of supported texture mapping modes.

- enum CRAY_TEXTURE_FILTERING { CTF_BOX, CTF_TRIANGLE }

  List of supported texture filtering modes.

5.4.1 Detailed Description

Defines the enums used in the engine to define groups of elements for various operations.

5.4.2 Typedef Documentation

5.4.2.1 CRAY_ACCELERATION_STRUCTURE

```
typedef enum CRAY_ACCELERATION_STRUCTURE CRAY_ACCELERATION_STRUCTURE
```

List of supported acceleration structures.

5.4.2.2 CRAY_INTEGRATOR

```
typedef enum CRAY_INTEGRATOR CRAY_INTEGRATOR
```

List of supported integrators.

5.4.2.3 CRAY_IOR_CONDUCTORS

```
typedef enum CRAY_IOR_CONDUCTORS CRAY_IOR_CONDUCTORS
```

List of supported conductors (IOR and absorption), sampled at wavelengths 645nm, 526nm, 444nm. Source: https://refractiveindex.info.

5.4.2.4 CRAY_IOR_DIELECTRICS

```
typedef enum CRAY_IOR_DIELECTRICS CRAY_IOR_DIELECTRICS
```

List of IOR for supported dielectrics, sampled at wavelength 526nm. Source: https://refractiveindex.↩info.

5.4.2.5 CRAY_LOGGERENTRY

```
typedef enum CRAY_LOGGERENTRY CRAY_LOGGERENTRY
```

List of supported log messages.

5.4.2.6 CRAY_TEXTURE_FILTERING

```
typedef enum CRAY_TEXTURE_FILTERING CRAY_TEXTURE_FILTERING
```

List of supported texture filtering modes.

5.4.2.7 CRAY_TEXTURE_MAPPING

```
typedef enum CRAY_TEXTURE_MAPPING CRAY_TEXTURE_MAPPING
```

List of supported texture mapping modes.

5.4.2.8 CRAY_TEXTURE_WRAPPING

```
typedef enum CRAY_TEXTURE_WRAPPING CRAY_TEXTURE_WRAPPING
```

List of supported texture wrapping modes.

5.4.3 Enumeration Type Documentation

5.4.3.1 CRAY_ACCELERATION_STRUCTURE

```
enum CRAY_ACCELERATION_STRUCTURE
```

List of supported acceleration structures.

Enumerator

CR_AS_ARRAY	Array-based. No acceleration structure.
CR_AS_BVH	Bounding Volume Hierarchy.
CR_AS_NONE	Unused value.

5.4.3.2 CRAY_INTEGRATOR

enum CRAY_INTEGRATOR

List of supported integrators.

Enumerator

CR_INTEGRATOR_PT	Unidirectional path tracer.
CR_INTEGRATOR_AO	Ambient Occlusion.
CR_INTEGRATOR_NONE	Unused value.

5.4.3.3 CRAY_IOR_CONDUCTORS

enum CRAY_IOR_CONDUCTORS

List of supported conductors (IOR and absorption), sampled at wavelengths 645nm, 526nm, 444nm. Source: https://refractiveindex.info.

Enumerator

CR_CONDUCTOR_Alluminium_Al	
CR_CONDUCTOR_Brass_CuZn	Alluminium.
CR_CONDUCTOR_Copper_Cu	Brass.
CR_CONDUCTOR_Gold_Au	Copper.
CR_CONDUCTOR_Iron_Fe	Gold.
CR_CONDUCTOR_Silver_Ag	Iron.
CR_CONDUCTOR_ALL_CONDUCTORS	Silver.

5.4.3.4 CRAY_IOR_DIELECTRICS

enum CRAY_IOR_DIELECTRICS

List of IOR for supported dielectrics, sampled at wavelength 526nm. Source: https://refractiveindex.↩info.

Enumerator

CR_DIELECTRIC_Acrylic_glass	
CR_DIELECTRIC_Polystyrene	Plastics: Poly(methyl methacrylate)
CR_DIELECTRIC_Polycarbonate	Plastics: Polystyrene.
CR_DIELECTRIC_Diamond	Plastics: Polycarbonate.

Enumerator

CR_DIELECTRIC_Ice	Crystals: Diamond.
CR_DIELECTRIC_Sapphire	Crystals: Ice.
CR_DIELECTRIC_Crown_Glass_bk7	Crystals: Sapphire.
CR_DIELECTRIC_Soda_lime_glass	
CR_DIELECTRIC_Water_25C	
CR_DIELECTRIC_Acetone_20C	Liquids: Water at 25C.
CR_DIELECTRIC_Air	Liquids: Acetone at 20C.
CR_DIELECTRIC_Carbon_dioxide	Gasses: Air.
CR_DIELECTRIC_ALL_DIELECTRICS	Gasses: Carbon dioxide. Unused value

5.4.3.5 CRAY_LOGGERENTRY

`enum CRAY_LOGGERENTRY`

List of supported log messages.

Enumerator

CR_LOGGER_ERROR	Error messages: Execution will probably stop.
CR_LOGGER_WARNING	Warning messages: Some operation was unexpected or an error was caught and handled properly.
CR_LOGGER_INFO	Info messages: General information. Stats, timings, etc.
CR_LOGGER_ASSERT	Asserts: Something that shouldn't happen, happened.
CR_LOGGER_DEBUG	Debug: Verbose info. This should only be used in extreme cases or in conjunction with CR_debug_setTestPixelRange.
CR_LOGGER_NOTHING	Unused value.

5.4.3.6 CRAY_TEXTURE_FILTERING

`enum CRAY_TEXTURE_FILTERING`

List of supported texture filtering modes.

Enumerator

CTF_BOX	Nearest neighbor, i.e. selecting the closest sample.
CTF_TRIANGLE	Bilinear filtering, i.e. selecting the average of four.

5.4.3.7 CRAY_TEXTURE_MAPPING

`enum CRAY_TEXTURE_MAPPING`

List of supported texture mapping modes.

Enumerator

CTM_UV	The primitive's uvs are used.
CTM_PLANAR	Planar mapping.
CTM_SPHERICAL	Spherical mapping.

5.4.3.8 CRAY_TEXTURE_WRAPPING

`enum CRAY_TEXTURE_WRAPPING`

List of supported texture wrapping modes.

Enumerator

CTW_CLAMP	UVs are clamped to [0, 1].
CTW_REPEAT	UVs are repeated, i.e. using the fractional part only.

5.5 General Settings API

General functions setting/retrieves internal engine state.

Functions

- void CR_enableRGBBuffer (bool enable)

  Enables/disables the use of an RGB buffer. By default, all results are written to a floating buffer, for increased precision. When an RGB buffer is enabled, the main writing is still performed on the floating buffer. The RGB buffer simply contains the converted 8-bit values after tonemapping and gamma correction. This is useful mainly for GUI operations, in order to avoid performing a whole conversion later on.
- void CR_setOutputName (const char *name)

  Sets an output name for writing. This is just a utility function, e.g. for scene loaders, as the engine does not perform any file operations. The name does not contain any extension. It is the responsibility of the caller to retrieve the final framebuffer and write to whichever file format is desired (or do anything with it).
- void CR_setFilmDimensions (int32_t width, int32_t height)

  Sets the image dimensions.
- void CR_setNumThreads (int32_t num_threads)

  Sets the number of threads to use for rendering. Ideally, this should be the number of cores in the system. The actual number of threads spawn is num_threads - 1 unless the use_main_thread flag is set to false. A value of 0 issues a serial operation.
- void CR_useMainThread (bool use_main_thread)

  Enables/disables the use of the main thread.
- void CR_setBlockSize (int32_t block_size)

  Sets the number of pixels per 2D tile to use for multithreaded rendering. A value of 16 or 32 is a typical use case.
- void CR_setSamplesPerPixel (int32_t spp)

  Sets the number of ray samples per pixel.
- void CR_setGamma (cr_float gamma)

  Sets the gamma correction value. This is used only when an RGB buffer is enabled.
- void CR_setExposure (cr_float exposure)

  Sets the exposure value to be used during the tonemapping operation. Internally, the traditional John Hable's Uncharted 2 tonemapping operator is used. This is used only when an RGB buffer is enabled. Using an exposure value of 0.0 degrades to no tonemapping. This process is then followed by gamma correction (see CR_setGamma)
- void CR_setAccelerationStructure (CRAY_ACCELERATION_STRUCTURE acceleration_structure_type)

  Sets the acceleration structure. Simple scenes can use a simple array structure. For anything else, a BVH is the optimal option. Any subsequent calls to this function, overrides AND DELETES any previously allocated acceleration structure and internal data, e.g. primitives. This function should be called before adding any primitives.
- void CR_setIntegrator (CRAY_INTEGRATOR integrator_type)

  Adds an integrator. Currently, Ambient Occlusion and unidirectional Path Tracing are available. A scene can be rendered multiple times with different integrators, by simply issuing calls to this function and CR_start, i.e. no reloading of the scene data is required.
- bool CR_setInteractiveRender (bool enable)

  Enables/disables interactive rendering. This is useful only for integrations with a GUI. Enabling interactive rendering maintains all current settings with the following differences:
- void CR_useFastRNG (bool enable)

  Enforces the use of Xorshift RNG.
- bool CR_isRGBBufferEnabled ()

  Check if RGB buffer is enabled.
- const char * CR_getOutputName ()

  Gets the output name.
- float * CR_getFilmFloatDataPtr ()

  Gets a pointer to the internal float framebuffer. This should be called when rendering has finished, e.g. to write to a file.

- uint64_t CR_getFilmFloatByteSize ()

  Gets the number of bytes for the float buffer.
- uint8_t * CR_getFilmRGBDataPtr ()

  Gets a pointer to the internal RGB framebuffer. This should be called when rendering has finished, e.g. to write to a file.
- uint64_t CR_getFilmRGBByteSize ()

  Gets the number of bytes for the RGB buffer.
- int32_t CR_getFilmWidth ()

  Gets the width of the internal framebuffer.
- int32_t CR_getFilmHeight ()

  Gets the height of the internal framebuffer.
- int32_t CR_getFilmChannels ()

  Gets the number of channels of the internal framebuffer. This is currently fixed to 3.
- int32_t CR_getNumThreads ()

  Gets the number of threads.
- bool CR_usesMainThread ()

  Check if main thread rendering is enabled.
- int32_t CR_getBlockSize ()

  Gets the number of pixels per 2D tile.
- int32_t CR_getSamplesPerPixel ()

  Gets the number of ray samples per pixel.
- cr_float CR_getGamma ()

  Gets the gamma correction value.
- cr_float CR_getExposure ()

  Gets the exposure value used during the tonemapping operation.
- CRAY_ACCELERATION_STRUCTURE CR_getAccelerationStructure ()

  Gets the acceleration structure type used.
- CRAY_INTEGRATOR CR_getIntegrator ()

  Gets the integrator type used.
- bool CR_isInteractiveRendeEnabled ()

  Check if interactive rendering is enabled.
- bool CR_isFastRNGEnabled ()

  Check if Fast RNG is enabled.
- void CR_setAOSamplesPerPixel (int32_t spp)

  Sets the number of AO rays to spawn for each sample per pixel.
- void CR_setAORange (cr_float range)

  Sets the AO range of the rays. A value of 0 denotes infinite range.
- int32_t CR_getAOSamplesPerPixel ()

  Gets the number of AO rays spawned for each sample per pixel.
- cr_float CR_getAORange ()

  Gets the AO range of the rays.
- void CR_setRayDepth (int32_t depth)

  Sets the maximum depth of the rays for the path tracer. 1 is direct lighting, 2 is 1 indirect bounce, etc.
- void CR_setRussianRoulette (bool enable)

  Enables/disables russian roulette for path tracing. This function is also capped by the CR_setRayDepth.
- int32_t CR_getRayDepth ()

  Gets the maximum depth of the rays for the path tracer.
- bool CR_getRussianRoulette ()

  Gets the russian roulette status.

5.5.1 Detailed Description

General functions setting/retrieves internal engine state.

5.5.2 Function Documentation

5.5.2.1 CR_enableRGBBuffer()

```
void CR_enableRGBBuffer (
            bool enable )
```

Enables/disables the use of an RGB buffer. By default, all results are written to a floating buffer, for increased precision. When an RGB buffer is enabled, the main writing is still performed on the floating buffer. The RGB buffer simply contains the converted 8-bit values after tonemapping and gamma correction. This is useful mainly for GUI operations, in order to avoid performing a whole conversion later on.

Parameters

enable	A flag to enable/disable the RGB buffer (Default: false).

5.5.2.2 CR_getAccelerationStructure()

```
CRAY_ACCELERATION_STRUCTURE CR_getAccelerationStructure ( )
```

Gets the acceleration structure type used.

Returns

The acceleration structure type.

5.5.2.3 CR_getAORange()

```
cr_float CR_getAORange ( )
```

Gets the AO range of the rays.

Returns

The AO range.

5.5.2.4 CR_getAOSamplesPerPixel()

```
int32_t CR_getAOSamplesPerPixel ( )
```

Gets the number of AO rays spawned for each sample per pixel.

Returns

The AO rays per spp.

5.5.2.5 CR_getBlockSize()

```
int32_t CR_getBlockSize ( )
```

Gets the number of pixels per 2D tile.

Returns

The number of pixels per 2D tile.

5.5.2.6 CR_getExposure()

```
cr_float CR_getExposure ( )
```

Gets the exposure value used during the tonemapping operation.

5.5.2.7 CR_getFilmChannels()

```
int32_t CR_getFilmChannels ( )
```

Gets the number of channels of the internal framebuffer. This is currently fixed to 3.

Returns

The number of channels.

5.5.2.8 CR_getFilmFloatByteSize()

```
uint64_t CR_getFilmFloatByteSize ( )
```

Gets the number of bytes for the float buffer.

Returns

The number of bytes.

5.5.2.9 CR_getFilmFloatDataPtr()

```
float* CR_getFilmFloatDataPtr ( )
```

Gets a pointer to the internal float framebuffer. This should be called when rendering has finished, e.g. to write to a file.

Returns

A pointer to the data.

5.5.2.10 CR_getFilmHeight()

```
int32_t CR_getFilmHeight ( )
```

Gets the height of the internal framebuffer.

Returns

The height in pixels.

5.5.2.11 CR_getFilmRGBByteSize()

```
uint64_t CR_getFilmRGBByteSize ( )
```

Gets the number of bytes for the RGB buffer.

Returns

The number of bytes.

5.5.2.12 CR_getFilmRGBDataPtr()

```
uint8_t* CR_getFilmRGBDataPtr ( )
```

Gets a pointer to the internal RGB framebuffer. This should be called when rendering has finished, e.g. to write to a file.

Returns

A pointer to the data.

5.5.2.13 CR_getFilmWidth()

```
int32_t CR_getFilmWidth ( )
```

Gets the width of the internal framebuffer.

Returns

The width in pixels.

5.5.2.14 CR_getGamma()

```
cr_float CR_getGamma ( )
```

Gets the gamma correction value.

Returns

The gamma correction value.

5.5.2.15 CR_getIntegrator()

```
CRAY_INTEGRATOR CR_getIntegrator ( )
```

Gets the integrator type used.

Returns

The integrator type.

5.5.2.16 CR_getNumThreads()

```
int32_t CR_getNumThreads ( )
```

Gets the number of threads.

Returns

The number of threads.

5.5.2.17 CR_getOutputName()

```
const char* CR_getOutputName ( )
```

Gets the output name.

Returns

The output name.

5.5.2.18 CR_getRayDepth()

```
int32_t CR_getRayDepth ( )
```

Gets the maximum depth of the rays for the path tracer.

Returns

The maximum depth of rays.

5.5.2.19 CR_getRussianRoulette()

```
bool CR_getRussianRoulette ( )
```

Gets the russian roulette status.

Returns

Returns true if russian roulette is enabled, false otherwise.

5.5.2.20 CR_getSamplesPerPixel()

```
int32_t CR_getSamplesPerPixel ( )
```

Gets the number of ray samples per pixel.

Returns

The number of ray samples per pixel.

5.5.2.21 CR_isFastRNGEnabled()

```
bool CR_isFastRNGEnabled ( )
```

Check if Fast RNG is enabled.

Returns

True if Fast RNG is enabled/false otherwise.

5.5.2.22 CR_isInteractiveRendeEnabled()

```
bool CR_isInteractiveRendeEnabled ( )
```

Check if interactive rendering is enabled.

Returns

True if interactive rendering is enabled/false otherwise.

5.5.2.23 CR_isRGBBufferEnabled()

```
bool CR_isRGBBufferEnabled ( )
```

Check if RGB buffer is enabled.

Returns

True if RGB buffer is enabled/false otherwise.

5.5.2.24 CR_setAccelerationStructure()

```
void CR_setAccelerationStructure (
            CRAY_ACCELERATION_STRUCTURE acceleration_structure_type )
```

Sets the acceleration structure. Simple scenes can use a simple array structure. For anything else, a BVH is the optimal option. Any subsequent calls to this function, overrides AND DELETES any previously allocated acceleration structure and internal data, e.g. primitives. This function should be called before adding any primitives.

Parameters

acceleration_structure_type	The acceleration structure (Default: BVH).

5.5.2.25 CR_setAORange()

```
void CR_setAORange (
            cr_float range )
```

Sets the AO range of the rays. A value of 0 denotes infinite range.

Parameters

range	The AO range (Default: 0).

5.5.2.26 CR_setAOSamplesPerPixel()

```
void CR_setAOSamplesPerPixel (
            int32_t spp )
```

Sets the number of AO rays to spawn for each sample per pixel.

Parameters

spp	The AO rays per spp (Default: 1).

5.5.2.27 CR_setBlockSize()

```
void CR_setBlockSize (
            int32_t block_size )
```

Sets the number of pixels per 2D tile to use for multithreaded rendering. A value of 16 or 32 is a typical use case.

Parameters

block_size	The number of tiles (Default: 16, Minimum: 16).

5.5.2.28 CR_setExposure()

```
void CR_setExposure (
            cr_float exposure )
```

Sets the exposure value to be used during the tonemapping operation. Internally, the traditional John Hable's Uncharted 2 tonemapping operator is used. This is used only when an RGB buffer is enabled. Using an exposure value of 0.0 degrades to no tonemapping. This process is then followed by gamma correction (see CR_setGamma)

Parameters

exposure	The exposure value (Default: 2.0).

5.5.2.29 CR_setFilmDimensions()

```
void CR_setFilmDimensions (
            int32_t width,
            int32_t height )
```

Sets the image dimensions.

Parameters

width	The width in pixels (Minimum: 32).
height	The height in pixels (Minimum: 32).

5.5.2.30 CR_setGamma()

```
void CR_setGamma (
            cr_float gamma )
```

Sets the gamma correction value. This is used only when an RGB buffer is enabled.

Parameters

gamma	The samples per pixel (Default: 2.2).

5.5.2.31 CR_setIntegrator()

```
void CR_setIntegrator (
            CRAY_INTEGRATOR integrator_type )
```

Adds an integrator. Currently, Ambient Occlusion and unidirectional Path Tracing are available. A scene can be rendered multiple times with different integrators, by simply issuing calls to this function and CR_start, i.e. no reloading of the scene data is required.

Parameters

integrator_type	The integrator to use (Default: Path Tracing).

5.5.2.32 CR_setInteractiveRender()

```
bool CR_setInteractiveRender (
            bool enable )
```

Enables/disables interactive rendering. This is useful only for integrations with a GUI. Enabling interactive rendering maintains all current settings with the following differences:

- Rendering is using 1spp and the seed is restarted in every iteration. Therefore this mode is for preview purposes only.
- All system cores are utilized (minus the main thread).
- The RNG generator used is a classic Xorshift RNG instead of mt19937. Note that interactive rendering cannot be enabled when rendering in the main thread is also enabled (see CR_setNumThreads).

Parameters

enable	A flag to enable/disable interactive rendering (Default: false).

Returns

returns true if interactive mode is enabled, false otherwise.

5.5.2.33 CR_setNumThreads()

```
void CR_setNumThreads (
            int32_t num_threads )
```

Sets the number of threads to use for rendering. Ideally, this should be the number of cores in the system. The actual number of threads spawn is num_threads - 1 unless the use_main_thread flag is set to false. A value of 0 issues a serial operation.

Parameters

num_threads	The number of threads to spawn (Default: 1).

5.5.2.34 CR_setOutputName()

```
void CR_setOutputName (
            const char * name )
```

Sets an output name for writing. This is just a utility function, e.g. for scene loaders, as the engine does not perform any file operations. The name does not contain any extension. It is the responsibility of the caller to retrieve the final framebuffer and write to whichever file format is desired (or do anything with it).

Parameters

name	The output name (Default: result).

5.5.2.35 CR_setRayDepth()

```
void CR_setRayDepth (
            int32_t depth )
```

Sets the maximum depth of the rays for the path tracer. 1 is direct lighting, 2 is 1 indirect bounce, etc.

Parameters

depth	The maximum depth of rays (Default: 10).

5.5.2.36 CR_setRussianRoulette()

```
void CR_setRussianRoulette (
            bool enable )
```

Enables/disables russian roulette for path tracing. This function is also capped by the CR_setRayDepth.

Parameters

enable	A flag to enable/disable russian roulette (Default: true).

5.5.2.37 CR_setSamplesPerPixel()

```
void CR_setSamplesPerPixel (
            int32_t spp )
```

Sets the number of ray samples per pixel.

Parameters

spp	The samples per pixel (Default: 1).

5.5.2.38 CR_useFastRNG()

```
void CR_useFastRNG (
            bool enable )
```

Enforces the use of Xorshift RNG.

Parameters

enable	A flag to enable/disable fast RNG (Default: false).

5.5.2.39 CR_useMainThread()

```
void CR_useMainThread (
            bool use_main_thread )
```

Enables/disables the use of the main thread.

Parameters

use_main_thread	A flag to enforce the use of the main thread. This should be disabled for GUI integration (Default: true).

5.5.2.40 CR_usesMainThread()

```
bool CR_usesMainThread ( )
```

Check if main thread rendering is enabled.

Returns

True if the main thread is used during rendering, false otherwise.

5.6 Generic Resource Handling API

Base engine functions for rendering operations and general resource handling.

Functions

- void CR_init ()

  Initializes the rendering engine. This must be the first function call.
- void CR_destroy ()

  Destroys any internal allocations. This is only necessary if multiple rendering operations are to be performed.
- void CR_start ()

  Starts the rendering operation: performs scene preprocessing, initializes threads and starts rendering.
- void CR_stop ()

  Issues a request to stop any running rendering operations.
- void CR_clearScene ()

  Clears any internal scene data, e.g. cameras, acceleration structures, lights, materials, etc. This is needed only if a subsequent rendering operation is to be performed.
- bool CR_started ()

  Queries the renderer if a start command has been issued.
- bool CR_finished ()

  Queries the renderer if rendering has finished.

5.6.1 Detailed Description

Base engine functions for rendering operations and general resource handling.

5.6.2 Function Documentation

5.6.2.1 CR_clearScene()

```
void CR_clearScene ( )
```

Clears any internal scene data, e.g. cameras, acceleration structures, lights, materials, etc. This is needed only if a subsequent rendering operation is to be performed.

5.6.2.2 CR_destroy()

```
void CR_destroy ( )
```

Destroys any internal allocations. This is only necessary if multiple rendering operations are to be performed.

5.6.2.3 CR_finished()

```
bool CR_finished ( )
```

Queries the renderer if rendering has finished.

Returns

Returns true if the renderer has finished, false otherwise.

5.6.2.4 CR_init()

```
void CR_init ( )
```

Initializes the rendering engine. This must be the first function call.

5.6.2.5 CR_start()

```
void CR_start ( )
```

Starts the rendering operation: performs scene preprocessing, initializes threads and starts rendering.

5.6.2.6 CR_started()

```
bool CR_started ( )
```

Queries the renderer if a start command has been issued.

Returns

Returns true if the renderer is doing work, false otherwise.

5.6.2.7 CR_stop()

```
void CR_stop ( )
```

Issues a request to stop any running rendering operations.

5.7 Image API

Functions for the generation of images, samplers and textures (an image with a sampler).

Functions

- CRAY_HANDLE CR_addImage (const uint8_t *ptr, int32_t width, int32_t height, int32_t channels, int32_t size_type, bool is_srgb)

  Creates an image. The data is internally allocated.
- CRAY_HANDLE CR_addImageCheckerboard (cr_vec3 tex1_rgb, cr_vec3 tex2_rgb, cr_float frequency)

  Creates a checkerboard texture using two different colors.
- CRAY_HANDLE CR_addTextureSampler (CRAY_TEXTURE_MAPPING mapping_mode, CRAY_TEXTURE_WRAPPING wrap_mode, CRAY_TEXTURE_FILTERING filtering_mode)

  Creates a sampler.
- CRAY_HANDLE CR_addTexture (CRAY_HANDLE image_id, CRAY_HANDLE sampler_id)

  Associates a texture with a sampler to be used for rendering.
- void CR_addMaterialDiffuseReflectionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's diffuse reflection color.
- void CR_addMaterialDiffuseTransmissionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's diffuse transmission color.
- void CR_addMaterialSpecularReflectionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's specular reflection color.
- void CR_addMaterialSpecularTransmissionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's specular transmission color.

5.7.1 Detailed Description

Functions for the generation of images, samplers and textures (an image with a sampler).

5.7.2 Function Documentation

5.7.2.1 CR_addImage()

```
CRAY_HANDLE CR_addImage (
            const uint8_t * ptr,
            int32_t width,
            int32_t height,
            int32_t channels,
            int32_t size_type,
            bool is_srgb )
```

Creates an image. The data is internally allocated.

Parameters

ptr	The pointer to the data
width	The width of the image
height	The height of the image
channels	The number of channels in the image
size_type	The size of the data type, e.g. 1 for RGB, 4 for floats
is_srgb	If the data should be converted to linear space during storage

Returns

A generic handle used to pass the image to other functions.

5.7.2.2 CR_addImageCheckerboard()

```
CRAY_HANDLE CR_addImageCheckerboard (
            cr_vec3 tex1_rgb,
            cr_vec3 tex2_rgb,
            cr_float frequency )
```

Creates a checkerboard texture using two different colors.

Parameters

tex1_rgb	The color of the texture for the odd texels
tex2_rgb	The color of the texture for the even texels
frequency	The repetition frequency per the uv range [0-1], e.g, a frequency of 1 shows 1 tile, a frequency of 2 shows 4. (Minimum value: 1.0)

Returns

A generic handle used to pass the image to other functions.

5.7.2.3 CR_addMaterialDiffuseReflectionTexture()

```
void CR_addMaterialDiffuseReflectionTexture (
            CRAY_HANDLE material_id,
            CRAY_HANDLE texture_id )
```

Associates a texture with a material's diffuse reflection color.

Parameters

material_id	The handle of the diffuse material.
texture_id	The handle of the texture.

5.7.2.4 CR_addMaterialDiffuseTransmissionTexture()

```
void CR_addMaterialDiffuseTransmissionTexture (
            CRAY_HANDLE material_id,
            CRAY_HANDLE texture_id )
```

Associates a texture with a material's diffuse transmission color.

Parameters

material_id	The handle of the diffuse material.
texture_id	The handle of the texture.

5.7.2.5 CR_addMaterialSpecularReflectionTexture()

```
void CR_addMaterialSpecularReflectionTexture (
            CRAY_HANDLE material_id,
            CRAY_HANDLE texture_id )
```

Associates a texture with a material's specular reflection color.

Parameters

material_id	The handle of the specular material.
texture_id	The handle of the texture.

5.7.2.6 CR_addMaterialSpecularTransmissionTexture()

```
void CR_addMaterialSpecularTransmissionTexture (
            CRAY_HANDLE material_id,
            CRAY_HANDLE texture_id )
```

Associates a texture with a material's specular transmission color.

Parameters

material_id	The handle of the specular material.
texture_id	The handle of the texture.

5.7.2.7 CR_addTexture()

```
CRAY_HANDLE CR_addTexture (
            CRAY_HANDLE image_id,
            CRAY_HANDLE sampler_id )
```

Associates a texture with a sampler to be used for rendering.

Parameters

image_id	The handle of the image.
sampler_id	The handle of the sampler.

Returns

A generic handle used to pass the texture to other functions.

5.7.2.8 CR_addTextureSampler()

```
CRAY_HANDLE CR_addTextureSampler (
            CRAY_TEXTURE_MAPPING mapping_mode,
            CRAY_TEXTURE_WRAPPING wrap_mode,
            CRAY_TEXTURE_FILTERING filtering_mode )
```

Creates a sampler.

Parameters

mapping_mode	The mapping mode used. (Default: UV).
wrap_mode	The wrap mode used. (Default: CLAMP).
filtering_mode	The filtering mode used. (Default: TRIANGLE/bilinear).

Returns

A generic handle used to pass the sampler to other functions.

5.8 Light API

Functions for the generation of emitters.

Functions

- void CR_addLightPointSpot (cr_vec3 position, cr_vec3 target, cr_vec3 intensity, cr_float falloffAngle, cr_float maxAngle, cr_float exponent)

  Creates a point spotlight. This is a dirac light.
- void CR_addLightPointOmni (cr_vec3 position, cr_vec3 intensity)

  Creates a point omnidirectional light. This is a dirac light.
- void CR_addLightDirectional (cr_vec3 direction, cr_vec3 intensity_rgb)

  Creates a parallel light using a direction.
- void CR_addLightDirectionalFromTo (cr_vec3 position, cr_vec3 target, cr_vec3 intensity_rgb)

  Creates a parallel light using a point and a target to generate a direction.
- CRAY_HANDLE CR_addLightAreaDiffuseEmitter (cr_vec3 flux, bool doublesided)

  Creates an area light, modeled as a diffuse emitter. This needs to be associated with a primitive. Currently, triangles are not yet supported.
- void CR_addAreaLightToPrimitive (CRAY_HANDLE primitive_id, CRAY_HANDLE light_id)

  Associates an area light with a primitive. Note that the same light source can be shared by multiple primitives. In that case, the light's flux is scaled by each primitive's area. This way, it is easy to associate a large polygon group with one light source. Since this method is approximate, accurate rendering requires 1 light per primitive.

5.8.1 Detailed Description

Functions for the generation of emitters.

5.8.2 Function Documentation

5.8.2.1 CR_addAreaLightToPrimitive()

```
void CR_addAreaLightToPrimitive (
            CRAY_HANDLE primitive_id,
            CRAY_HANDLE light_id )
```

Associates an area light with a primitive. Note that the same light source can be shared by multiple primitives. In that case, the light's flux is scaled by each primitive's area. This way, it is easy to associate a large polygon group with one light source. Since this method is approximate, accurate rendering requires 1 light per primitive.

Parameters

primitive↩ _id	The handle of the primitive
light_id	The handle of the light

5.8.2.2 CR_addLightAreaDiffuseEmitter()

```
CRAY_HANDLE CR_addLightAreaDiffuseEmitter (
            cr_vec3 flux,
            bool doublesided )
```

Creates an area light, modeled as a diffuse emitter. This needs to be associated with a primitive. Currently, triangles are not yet supported.

Parameters

flux	The flux of the light, in Watts.
doublesided	A flag that dictates if the light emits light from both its sides.

Returns

A generic handle used to pass the light to other functions.

5.8.2.3 CR_addLightDirectional()

```
void CR_addLightDirectional (
            cr_vec3 direction,
            cr_vec3 intensity_rgb )
```

Creates a parallel light using a direction.

Parameters

direction	The direction of the light.
intensity_rgb	The intensity of the light.

5.8.2.4 CR_addLightDirectionalFromTo()

```
void CR_addLightDirectionalFromTo (
            cr_vec3 position,
            cr_vec3 target,
            cr_vec3 intensity_rgb )
```

Creates a parallel light using a point and a target to generate a direction.

Parameters

position	The position of the light in world units.
target	The target of the light in world units.
intensity_rgb	The intensity of the light.

5.8.2.5 CR_addLightPointOmni()

```
void CR_addLightPointOmni (
            cr_vec3 position,
            cr_vec3 intensity )
```

Creates a point omnidirectional light. This is a dirac light.

Parameters

position	The position of the light in world units.
intensity	The intensity of the light.

5.8.2.6 CR_addLightPointSpot()

```
void CR_addLightPointSpot (
            cr_vec3 position,
            cr_vec3 target,
            cr_vec3 intensity,
            cr_float falloffAngle,
            cr_float maxAngle,
            cr_float exponent )
```

Creates a point spotlight. This is a dirac light.

Parameters

position	The position of the light in world units.
target	The target of the light in world units.
intensity	The intensity of the light.
falloffAngle	The falloff angle, that dictates where the light energy starts to drop.
maxAngle	The cutoff angle, that dictates where the light energy drops to 0.
exponent	The spotlight exponent, that provides an exponential fall off as the light travels away from its perfect direction.

5.9 Logging API

Functions for message logging and progress reporting.

Functions

- void CR_setMinimumLogLevel (CRAY_LOGGERENTRY type)

  Sets the minimum log level to store messages.
- CRAY_LOGGERENTRY CR_getMinimumLogLevel ()

  Gets the minimum log level.
- void CR_setLogFile (void *file)

  Enables writing to an output FILE. If disabled, the messages are written to a queue and can be polled using CR_getLastLogMessage.
- void * CR_getLogFile ()

  Gets the current output FILE.
- void CR_printTostdout (bool enable)

  Enables writing internal log messages. If disabled, the messages can be polled using CR_getLastLogMessage.
- bool CR_isPrintTostdoutEnabled ()

  Returns the status of printing to stdout.
- void CR_printProgressBar (bool enable)

  Enables of progress bar to stdout. If disabled, the progress percentage can be polled using CR_getProgressPercentage. Note that stdout printing must be enabled (see CR_printTostdout).
- bool CR_IsPrintProgressBarEnabled ()

  Returns the status of printing the progress bar to stdout. It is suggested to disable the progress bar when verbose logging is enabled, as it may interfere with the progress bar printing.
- cr_float CR_getProgressPercentage ()

  Retrieves the progress percentage of a running rendering process.
- bool CR_getLastLogMessage (char *msg, size_t *msg_length, CRAY_LOGGERENTRY *type)

  Gets the last log message stored by the rendering engine.

5.9.1 Detailed Description

Functions for message logging and progress reporting.

5.9.2 Function Documentation

5.9.2.1 CR_getLastLogMessage()

```
bool CR_getLastLogMessage (
            char * msg,
            size_t * msg_length,
            CRAY_LOGGERENTRY * type )
```

Gets the last log message stored by the rendering engine.

Parameters

msg	A preallocated buffer to store the message. A size of 1024 is a good choice.
msg_length	The actual length of the msg buffer. If the stored log message is larger, nothing is stored on the buffer and this value contains the required buffer size to use in a subsequent operation.
type	The logging message type

Returns

Returns true if a message has been successfully stored, false otherwise.

5.9.2.2 CR_getLogFile()

```
void* CR_getLogFile ( )
```

Gets the current output FILE.

Returns

Returns the FILE pointer.

5.9.2.3 CR_getMinimumLogLevel()

```
CRAY_LOGGERENTRY CR_getMinimumLogLevel ( )
```

Gets the minimum log level.

Returns

The minimum log level.

5.9.2.4 CR_getProgressPercentage()

```
cr_float CR_getProgressPercentage ( )
```

Retrieves the progress percentage of a running rendering process.

Returns

The progress percentage in the range [0-100]

5.9.2.5 CR_IsPrintProgressBarEnabled()

```
bool CR_IsPrintProgressBarEnabled ( )
```

Returns the status of printing the progress bar to stdout. It is suggested to disable the progress bar when verbose logging is enabled, as it may interfere with the progress bar printing.

Returns

True if printing the progress bar to stdout is enabled, false otherwise.

5.9.2.6 CR_isPrintTostdoutEnabled()

```
bool CR_isPrintTostdoutEnabled ( )
```

Returns the status of printing to stdout.

Returns

True if print to stdout is enabled, false otherwise.

5.9.2.7 CR_printProgressBar()

```
void CR_printProgressBar (
            bool enable )
```

Enables of progress bar to stdout. If disabled, the progress percentage can be polled using CR_getProgressPercentage. Note that stdout printing must be enabled (see CR_printTostdout).

Parameters

enable	A flag to enable/disable writing the rendering progress to stdout. (Default: true).

5.9.2.8 CR_printTostdout()

```
void CR_printTostdout (
            bool enable )
```

Enables writing internal log messages. If disabled, the messages can be polled using CR_getLastLogMessage.

Parameters

enable	A flag to enable/disable writing to stdout. (Default: true).

5.9.2.9 CR_setLogFile()

```
void CR_setLogFile (
            void * file )
```

Enables writing to an output FILE. If disabled, the messages are written to a queue and can be polled using CR_getLastLogMessage.

Parameters

file	A valid FILE pointer. (Default: NULL).

5.9.2.10 CR_setMinimumLogLevel()

```
void CR_setMinimumLogLevel (
            CRAY_LOGGERENTRY type )
```

Sets the minimum log level to store messages.

Parameters

type	The logger type. (Default: Error. Only error messages are logged).

5.10 Materials API

Functions for the creation and handling of materials.

Functions

- cr_float CR_getDielectricIOR (CRAY_IOR_DIELECTRICS dielectric)

  Returns the internal index of refraction from a list of predefined dielectrics.
- CRAY_HANDLE CR_addMaterialDiffuseReflection (cr_vec3 diffuse_reflection_color)

  Creates a diffuse reflection material, modeled as a pure lambertian surface. It can be used to model matte materials.
- CRAY_HANDLE CR_addMaterialDiffuseTransmission (cr_vec3 diffuse_transmission_color)

  Creates a diffuse transmission material, modeled as a pure lambertian surface. It can be used to model translucent materials.
- CRAY_HANDLE CR_addMaterialDiffuse (cr_vec3 diffuse_reflection_color, cr_vec3 diffuse_transmission_↩ color, cr_float eta_t)

  Creates a diffuse material representing rough reflection and transmission, modeled as a pure lambertian surface. It can be used to model translucent materials.
- cr_vec3 CR_getConductorIOR (CRAY_IOR_CONDUCTORS conductor)

  Returns the internal index of refraction from a list of predefined conductors.
- cr_vec3 CR_getConductorAbsorption (CRAY_IOR_CONDUCTORS conductor)

  Returns the absorption coefficients from a list of predefined conductors.
- CRAY_HANDLE CR_addMaterialConductor (cr_vec3 reflection_color, cr_vec3 absorption, cr_vec3 eta_↩ t, cr_float roughness)

  Creates a conductor material.
- CRAY_HANDLE CR_addMaterialDielectricSpecularReflection (cr_vec3 reflection_color, cr_float roughness)

  Creates a dielectric material representing smooth/rough reflection with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model mirror materials.
- CRAY_HANDLE CR_addMaterialDielectricSpecular (cr_vec3 reflection_color, cr_vec3 transmission_color, cr_float eta_t, cr_float roughness)

  Creates a dielectric material representing smooth/rough reflection and transmission with an isotropic Trowbridge-↩ Reitz(GGX) microfacet model. It can be used to model glass materials.
- CRAY_HANDLE CR_addMaterialDielectricSpecularTransmission (cr_vec3 transmission_color, cr_float eta↩ _t, cr_float roughness)

  Creates a dielectric material representing smooth/rough transmission with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model purely transmissive surfaces.
- CRAY_HANDLE CR_addMaterialDiffuseSpecularReflection (cr_vec3 diffuse_reflection_color, cr_vec3 specular_reflection_color, cr_float eta_t, cr_float roughness)

  Creates a mixed dielectric material with both diffuse and specular (both smooth/rough) reflection components. The diffuse component is modeled as a pure lambertian surface, while the specular employs an isotropic Trowbridge-↩ Reitz(GGX) microfacet model.

5.10.1 Detailed Description

Functions for the creation and handling of materials.

5.10.2 Function Documentation

5.10.2.1 CR_addMaterialConductor()

```
CRAY_HANDLE CR_addMaterialConductor (
             cr_vec3 reflection_color,
             cr_vec3 absorption,
             cr_vec3 eta_t,
             cr_float roughness )
```

Creates a conductor material.

Parameters

reflection_color	The reflection color of the component. This can be used for artistic purposes, as the material's physical color is dictated by the other parameters.
absorption	The absorption coefficients. This can be provided directly or through the list of predefined materials using CR_getConductorAbsorption.
eta_t	The index of refraction. This can be provided directly or through the list of predefined materials using CR_getConductorIOR.
roughness	The roughness parameter of the specular component. 0 denotes a perfectly smooth (dirac) surface.

Returns

A generic handle used to pass the material to other functions.

5.10.2.2 CR_addMaterialDielectricSpecular()

```
CRAY_HANDLE CR_addMaterialDielectricSpecular (
             cr_vec3 reflection_color,
             cr_vec3 transmission_color,
             cr_float eta_t,
             cr_float roughness )
```

Creates a dielectric material representing smooth/rough reflection and transmission with an isotropic Trowbridge-↩ Reitz(GGX) microfacet model. It can be used to model glass materials.

Parameters

reflection_color	The reflection color of the specular component.
transmission_color	The transmission color of the specular component.
eta_t	The index of refraction. This can be provided directly or through the list of predefined materials using CR_getDielectricIOR.
roughness	The roughness parameter of the specular component. 0 denotes a perfectly smooth (dirac) surface.

Returns

A generic handle used to pass the material to other functions.

5.10.2.3 CR_addMaterialDielectricSpecularReflection()

```
CRAY_HANDLE CR_addMaterialDielectricSpecularReflection (
            cr_vec3 reflection_color,
            cr_float roughness )
```

Creates a dielectric material representing smooth/rough reflection with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model mirror materials.

Parameters

reflection_color	The reflection color of the specular component.
roughness	The roughness parameter of the specular component. 0 denotes a perfectly smooth (dirac) surface.

Returns

A generic handle used to pass the material to other functions.

5.10.2.4 CR_addMaterialDielectricSpecularTransmission()

```
CRAY_HANDLE CR_addMaterialDielectricSpecularTransmission (
            cr_vec3 transmission_color,
            cr_float eta_t,
            cr_float roughness )
```

Creates a dielectric material representing smooth/rough transmission with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model purely transmissive surfaces.

Parameters

transmission_color	The transmission color of the specular component.
eta_t	The index of refraction. This can be provided directly or through the list of predefined materials using CR_getDielectricIOR.
roughness	The roughness parameter of the specular component. 0 denotes a perfectly smooth (dirac) surface.

Returns

A generic handle used to pass the material to other functions.

5.10.2.5 CR_addMaterialDiffuse()

```
CRAY_HANDLE CR_addMaterialDiffuse (
            cr_vec3 diffuse_reflection_color,
```

```
            cr_vec3 diffuse_transmission_color,
            cr_float eta_t )
```

Creates a diffuse material representing rough reflection and transmission, modeled as a pure lambertian surface. It can be used to model translucent materials.

Parameters

diffuse_reflection_color	The reflection color of the material.
diffuse_transmission_color	The transmission color of the material.
eta_t	The index of refraction. This can be provided directly or through the list of predefined materials using CR_getDielectricIOR.

5.10.2.6 CR_addMaterialDiffuseReflection()

```
CRAY_HANDLE CR_addMaterialDiffuseReflection (
            cr_vec3 diffuse_reflection_color )
```

Creates a diffuse reflection material, modeled as a pure lambertian surface. It can be used to model matte materials.

Parameters

diffuse_reflection_color	The reflection color of the material.

Returns

A generic handle used to pass the material to other functions.

5.10.2.7 CR_addMaterialDiffuseSpecularReflection()

```
CRAY_HANDLE CR_addMaterialDiffuseSpecularReflection (
            cr_vec3 diffuse_reflection_color,
            cr_vec3 specular_reflection_color,
            cr_float eta_t,
            cr_float roughness )
```

Creates a mixed dielectric material with both diffuse and specular (both smooth/rough) reflection components. The diffuse component is modeled as a pure lambertian surface, while the specular employs an isotropic Trowbridge-↩ Reitz(GGX) microfacet model.

Parameters

diffuse_reflection_color	The reflection color of the diffuse component.
specular_reflection_color	The reflection color of the specular component.
eta_t	The index of refraction. This can be provided directly or through the list of predefined materials using CR_getDielectricIOR.
roughness	The roughness parameter of the specular component. 0 denotes a perfectly smooth (dirac) surface.

Returns

A generic handle used to pass the material to other functions.

5.10.2.8 CR_addMaterialDiffuseTransmission()

```
CRAY_HANDLE CR_addMaterialDiffuseTransmission (
            cr_vec3 diffuse_transmission_color )
```

Creates a diffuse transmission material, modeled as a pure lambertian surface. It can be used to model translucent materials.

Parameters

diffuse_transmission_color	The transmission color of the material.

5.10.2.9 CR_getConductorAbsorption()

```
cr_vec3 CR_getConductorAbsorption (
            CRAY_IOR_CONDUCTORS conductor )
```

Returns the absorption coefficients from a list of predefined conductors.

Parameters

conductor	The conductor type.

Returns

The index of refraction.

5.10.2.10 CR_getConductorIOR()

```
cr_vec3 CR_getConductorIOR (
            CRAY_IOR_CONDUCTORS conductor )
```

Returns the internal index of refraction from a list of predefined conductors.

Parameters

conductor	The conductor type

Returns

The index of refraction

5.10.2.11 CR_getDielectricIOR()

```
cr_float CR_getDielectricIOR (
            CRAY_IOR_DIELECTRICS dielectric )
```

Returns the internal index of refraction from a list of predefined dielectrics.

Parameters

dielectric	The dielectric type.

Returns

The index of refraction.

5.11 Shape API

Functions for the generation of shapes. Both parametric and polygon group shapes are supported.

Functions

- void CR_reservePrimitives (size_t size)

  Reserves memory for the requested number of primitives.
- void CR_reserveMaterials (size_t size)

  Reserves memory for the requested number of materials.
- size_t CR_getNumPrimitives ()

  Retrieves the current number of primitives.
- size_t CR_getNumMaterials ()

  Retrieves the current number of materials.
- CRAY_HANDLE CR_addPrimitiveSphere (cr_vec3 position, cr_float radius, bool flip_normals, bool double↩ _sided, CRAY_HANDLE material_id)

  Creates a sphere primitive.
- CRAY_HANDLE CR_addPrimitiveRectangleXZ (cr_float min_x, cr_float max_x, cr_float min_z, cr_float max_z, cr_float offset, bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an axis-aligned rectangle primitive on the XZ axis.
- CRAY_HANDLE CR_addPrimitiveRectangleXY (cr_float min_x, cr_float max_x, cr_float min_y, cr_float max_y, cr_float offset, bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an axis-aligned rectangle primitive on the XY axis.
- CRAY_HANDLE CR_addPrimitiveRectangleYZ (cr_float min_y, cr_float max_y, cr_float min_z, cr_float max_z, cr_float offset, bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an axis-aligned rectangle primitive on the YZ axis.
- CRAY_HANDLE CR_addPrimitiveTriangle (bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an empty triangle primitive.
- void CR_addPrimitiveTriangleVertexPositions (CRAY_HANDLE triangle_id, int32_t index, cr_vec3 position)

  Associates a group of vertices with an existing triangle.
- void CR_addPrimitiveTriangleVertexNormals (CRAY_HANDLE triangle_id, int32_t index, cr_vec3 normal)

  Associates a group of normals with an existing triangle. If these are not provided, triangle plane normals are used.
- void CR_addPrimitiveTriangleVertexTexcoords (CRAY_HANDLE triangle_id, int32_t index, cr_vec3 texcoord)

  Associates a group of texcoord with an existing triangle.

5.11.1 Detailed Description

Functions for the generation of shapes. Both parametric and polygon group shapes are supported.

5.11.2 Function Documentation

5.11.2.1 CR_addPrimitiveRectangleXY()

```
CRAY_HANDLE CR_addPrimitiveRectangleXY (
              cr_float min_x,
              cr_float max_x,
              cr_float min_y,
              cr_float max_y,
              cr_float offset,
              bool flip_normals,
              bool double_sided,
              CRAY_HANDLE material_id )
```

Creates an axis-aligned rectangle primitive on the XY axis.

Parameters

min_x	The minimum value in the X axis of the rectangle.
max_x	The maximum value in the X axis of the rectangle.
min_y	The minimum value in the Y axis of the rectangle.
max_y	The maximum value in the Y axis of the rectangle.
offset	The offset of the rectangle in the Z axis.
flip_normals	A flag that dictates if the primitive's normals are flipped.
double_sided	A flag that dictates if the primitive is double sided.
material_id	The material id that will be used by the primitive.

Returns

A generic handle used to pass the primitive to other functions.

5.11.2.2 CR_addPrimitiveRectangleXZ()

```
CRAY_HANDLE CR_addPrimitiveRectangleXZ (
              cr_float min_x,
              cr_float max_x,
              cr_float min_z,
              cr_float max_z,
              cr_float offset,
              bool flip_normals,
              bool double_sided,
              CRAY_HANDLE material_id )
```

Creates an axis-aligned rectangle primitive on the XZ axis.

Parameters

min_x	The minimum value in the X axis of the rectangle.
max_x	The maximum value in the X axis of the rectangle.
min_z	The minimum value in the Z axis of the rectangle.
max_z	The maximum value in the Z axis of the rectangle.
offset	The offset of the rectangle in the Y axis.
flip_normals	A flag that dictates if the primitive's normals are flipped.
double_sided	A flag that dictates if the primitive is double sided.
material_id	The material id that will be used by the primitive.

Returns

A generic handle used to pass the primitive to other functions.

5.11.2.3 CR_addPrimitiveRectangleYZ()

```
CRAY_HANDLE CR_addPrimitiveRectangleYZ (
            cr_float min_y,
            cr_float max_y,
            cr_float min_z,
            cr_float max_z,
            cr_float offset,
            bool flip_normals,
            bool double_sided,
            CRAY_HANDLE material_id )
```

Creates an axis-aligned rectangle primitive on the YZ axis.

Parameters

min_y	The minimum value in the Y axis of the rectangle.
max_y	The maximum value in the Y axis of the rectangle.
min_z	The minimum value in the Z axis of the rectangle.
max_z	The maximum value in the Z axis of the rectangle.
offset	The offset of the rectangle in the X axis.
flip_normals	A flag that dictates if the primitive's normals are flipped.
double_sided	A flag that dictates if the primitive is double sided.
material_id	The material id that will be used by the primitive.

Returns

A generic handle used to pass the primitive to other functions.

5.11.2.4 CR_addPrimitiveSphere()

```
CRAY_HANDLE CR_addPrimitiveSphere (
            cr_vec3 position,
            cr_float radius,
            bool flip_normals,
            bool double_sided,
            CRAY_HANDLE material_id )
```

Creates a sphere primitive.

Parameters

position	The center of the sphere in world units.

Parameters

radius	The radius of the sphere in world units.
flip_normals	A flag that dictates if the primitive's normals are flipped.
double_sided	A flag that dictates if the primitive is double sided.
material_id	The material id that will be used by the primitive.

Returns

A generic handle used to pass the primitive to other functions.

5.11.2.5 CR_addPrimitiveTriangle()

```
CRAY_HANDLE CR_addPrimitiveTriangle (
            bool flip_normals,
            bool double_sided,
            CRAY_HANDLE material_id )
```

Creates an empty triangle primitive.

Parameters

flip_normals	A flag that dictates if the primitive's normals are flipped.
double_sided	A flag that dictates if the primitive is double sided.
material_id	The material id that will be used by the primitive.

Returns

A generic handle used to pass the primitive to other functions.

5.11.2.6 CR_addPrimitiveTriangleVertexNormals()

```
void CR_addPrimitiveTriangleVertexNormals (
            CRAY_HANDLE triangle_id,
            int32_t index,
            cr_vec3 normal )
```

Associates a group of normals with an existing triangle. If these are not provided, triangle plane normals are used.

Parameters

triangle_id	The triangle id that the vertices belong to.
index	The vertex index of the triangle.
normal	The vertex normals.

5.11.2.7 CR_addPrimitiveTriangleVertexPositions()

```
void CR_addPrimitiveTriangleVertexPositions (
            CRAY_HANDLE triangle_id,
            int32_t index,
            cr_vec3 position )
```

Associates a group of vertices with an existing triangle.

Parameters

triangle↩ _id	The triangle id that the vertices belong to.
index	The vertex index of the triangle.
position	The vertex position in world units.

5.11.2.8 CR_addPrimitiveTriangleVertexTexcoords()

```
void CR_addPrimitiveTriangleVertexTexcoords (
            CRAY_HANDLE triangle_id,
            int32_t index,
            cr_vec3 texcoord )
```

Associates a group of texcoord with an existing triangle.

Parameters

triangle↩ _id	The triangle id that the vertices belong to.
index	The vertex index of the triangle.
texcoord	The vertex texcoords.

5.11.2.9 CR_getNumMaterials()

```
size_t CR_getNumMaterials ( )
```

Retrieves the current number of materials.

Returns

The number of materials.

5.11.2.10 CR_getNumPrimitives()

```
size_t CR_getNumPrimitives ( )
```

Retrieves the current number of primitives.

Returns

The number of primitives.

5.11.2.11 CR_reserveMaterials()

```
void CR_reserveMaterials (
            size_t size )
```

Reserves memory for the requested number of materials.

Parameters

size	The number of materials.

5.11.2.12 CR_reservePrimitives()

```
void CR_reservePrimitives (
            size_t size )
```

Reserves memory for the requested number of primitives.

Parameters

size	The number of primitives.

5.12 Transformation API

Functions for hierarchical basic affine transformations.

Functions

- void CR_pushMatrix (cr_mat4 matrix)

  Pushes a transformation matrix onto the stack.
- void CR_popMatrix ()

  Pops a transformation matrix from the stack.
- cr_mat4 CR_rotate (cr_float degrees, cr_vec3 axis)

  Utility function to generate a rotation matrix along an arbitrary axis.
- cr_mat4 CR_translate (cr_vec3 translate)

  Utility function to generate a translation matrix.
- cr_mat4 CR_scale (cr_vec3 scale)

  Utility function to generate a scale matrix.
- cr_mat4 CR_mulMatrix (cr_mat4 matrix1, cr_mat4 matrix2)

  Utility function to multiply two matrices, M = M1 ∗ M2.

5.12.1 Detailed Description

Functions for hierarchical basic affine transformations.

5.12.2 Function Documentation

5.12.2.1 CR_mulMatrix()

```
cr_mat4 CR_mulMatrix (
            cr_mat4 matrix1,
            cr_mat4 matrix2 )
```

Utility function to multiply two matrices, M = M1 ∗ M2.

Parameters

matrix1	The first matrix.
matrix2	The second matrix.

Returns

The new matrix

5.12.2.2 CR_popMatrix()

```
void CR_popMatrix ( )
```

Pops a transformation matrix from the stack.

5.12.2.3 CR_pushMatrix()

```
void CR_pushMatrix (
            cr_mat4 matrix )
```

Pushes a transformation matrix onto the stack.

Parameters

matrix	The matrix to push.

5.12.2.4 CR_rotate()

```
cr_mat4 CR_rotate (
            cr_float degrees,
            cr_vec3 axis )
```

Utility function to generate a rotation matrix along an arbitrary axis.

Parameters

degrees	The angle of rotation, in degrees.
axis	The axis of rotation.

Returns

The new matrix

5.12.2.5 CR_scale()

```
cr_mat4 CR_scale (
            cr_vec3 scale )
```

Utility function to generate a scale matrix.

Parameters

scale	The scale vector.

Returns

The new matrix

5.12.2.6 CR_translate()

```
cr_mat4 CR_translate (
            cr_vec3 translate )
```

Utility function to generate a translation matrix.

Parameters

translate	The translation vector.

Returns

The new matrix

Chapter 6

Class Documentation

6.1 cr_mat4 Struct Reference

Struct for a 4x4 matrix.

`#include <CRay.h>`

Public Attributes

- cr_float x0
- cr_float y0
- cr_float z0
- cr_float w0
- cr_float x1
- cr_float y1
- cr_float z1
- cr_float w1
- cr_float x2
- cr_float y2
- cr_float z2
- cr_float w2
- cr_float x3
- cr_float y3
- cr_float z3
- cr_float w3

6.1.1 Detailed Description

Struct for a 4x4 matrix.

6.1.2 Member Data Documentation

6.1.2.1 w0

```
cr_float cr_mat4::w0
```

6.1.2.2 w1

```
cr_float cr_mat4::w1
```

6.1.2.3 w2

```
cr_float cr_mat4::w2
```

6.1.2.4 w3

```
cr_float cr_mat4::w3
```

6.1.2.5 x0

```
cr_float cr_mat4::x0
```

6.1.2.6 x1

```
cr_float cr_mat4::x1
```

6.1.2.7 x2

```
cr_float cr_mat4::x2
```

6.1.2.8 x3

```
cr_float cr_mat4::x3
```

6.1.2.9 y0

```
cr_float cr_mat4::y0
```

6.1.2.10 y1

```
cr_float cr_mat4::y1
```

6.1.2.11 y2

```
cr_float cr_mat4::y2
```

6.1.2.12 y3

```
cr_float cr_mat4::y3
```

6.1.2.13 z0

```
cr_float cr_mat4::z0
```

6.1.2.14 z1

```
cr_float cr_mat4::z1
```

6.1.2.15 z2

```
cr_float cr_mat4::z2
```

6.1.2.16 z3

```
cr_float cr_mat4::z3
```

The documentation for this struct was generated from the following file:

- Projects/CRay/CRay.h

6.2 cr_vec3 Struct Reference

Struct for a 3-dimensional vector.

```
#include <CRay.h>
```

Public Attributes

- cr_float x
- cr_float y
- cr_float z

6.2.1 Detailed Description

Struct for a 3-dimensional vector.

6.2.2 Member Data Documentation

6.2.2.1 x

```
cr_float cr_vec3::x
```

6.2.2.2 y

```
cr_float cr_vec3::y
```

6.2.2.3 z

```
cr_float cr_vec3::z
```

The documentation for this struct was generated from the following file:

- Projects/CRay/CRay.h

Chapter 7

File Documentation

7.1 Projects/CRay/CRay.h File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Classes

- struct cr_vec3

  Struct for a 3-dimensional vector.
- struct cr_mat4

  Struct for a 4x4 matrix.

Macros

- #define DOUBLE_PRECISION

  Define to set double precision floats as default.
- #define VEC3(_x, _y, _z) (cr_vec3){_x,_y,_z}

  Vec3 constructor from 3 values.
- #define VEC3_XYZ(_x) (cr_vec3){_x,_x,_x}

  Vec3 constructor from 1 value.
- #define VEC3_ARRAY(_x) (cr_vec3){_x[0],_x[1],_x[2]}

  Vec3 constructor from array.
- #define MAT4(_x0, _y0, _z0, _w0, _x1, _y1, _z1, _w1, _x2, _y2, _z2, _w2, _x3, _y3, _z3, _w3) (cr_mat4){↩ _x0, _y0, _z0, _w0, _x1, _y1, _z1, _w1, _x2, _y2, _z2, _w2, _x3, _y3, _z3, _w3}

  Mat4 constructor from 16 values.
- #define MAT4_IDENTITY (cr_mat4){1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1}

  Mat4 constructor for identity matrix.
- #define CRAY_HANDLE uint64_t

  Opaque handle.
- #define CRAY_INVALID_HANDLE 0u

  Opaque invalid handle.

Typedefs

- typedef double cr_float

  floating point value
- typedef struct cr_vec3 cr_vec3

  Struct for a 3-dimensional vector.
- typedef struct cr_mat4 cr_mat4

  Struct for a 4x4 matrix.
- typedef enum CRAY_ACCELERATION_STRUCTURE CRAY_ACCELERATION_STRUCTURE

  List of supported acceleration structures.
- typedef enum CRAY_INTEGRATOR CRAY_INTEGRATOR

  List of supported integrators.
- typedef enum CRAY_LOGGERENTRY CRAY_LOGGERENTRY

  List of supported log messages.
- typedef enum CRAY_IOR_DIELECTRICS CRAY_IOR_DIELECTRICS

  List of IOR for supported dielectrics, sampled at wavelength 526nm. Source: https://refractiveindex.info.
- typedef enum CRAY_IOR_CONDUCTORS CRAY_IOR_CONDUCTORS

  List of supported conductors (IOR and absorption), sampled at wavelengths 645nm, 526nm, 444nm. Source: https://refractiveindex.info.
- typedef enum CRAY_TEXTURE_WRAPPING CRAY_TEXTURE_WRAPPING

  List of supported texture wrapping modes.
- typedef enum CRAY_TEXTURE_MAPPING CRAY_TEXTURE_MAPPING

  List of supported texture mapping modes.
- typedef enum CRAY_TEXTURE_FILTERING CRAY_TEXTURE_FILTERING

  List of supported texture filtering modes.

Enumerations

- enum CRAY_ACCELERATION_STRUCTURE { CR_AS_ARRAY, CR_AS_BVH, CR_AS_NONE }

  List of supported acceleration structures.
- enum CRAY_INTEGRATOR { CR_INTEGRATOR_PT, CR_INTEGRATOR_AO, CR_INTEGRATOR_NONE }

  List of supported integrators.
- enum CRAY_LOGGERENTRY { CR_LOGGER_ERROR, CR_LOGGER_WARNING, CR_LOGGER_INFO, CR_LOGGER_ASSERT, CR_LOGGER_DEBUG, CR_LOGGER_NOTHING }

  List of supported log messages.
- enum CRAY_IOR_DIELECTRICS { CR_DIELECTRIC_Acrylic_glass, CR_DIELECTRIC_Polystyrene, CR_DIELECTRIC_Polycarbonate, CR_DIELECTRIC_Diamond, CR_DIELECTRIC_Ice, CR_DIELECTRIC_Sapphire, CR_DIELECTRIC_Crown_Glass_bk7, CR_DIELECTRIC_Soda_lime_gla CR_DIELECTRIC_Water_25C, CR_DIELECTRIC_Acetone_20C, CR_DIELECTRIC_Air, CR_DIELECTRIC_Carbon_dioxide, CR_DIELECTRIC_ALL_DIELECTRICS }

  List of IOR for supported dielectrics, sampled at wavelength 526nm. Source: https://refractiveindex.info.
- enum CRAY_IOR_CONDUCTORS { CR_CONDUCTOR_Alluminium_Al, CR_CONDUCTOR_Brass_CuZn, CR_CONDUCTOR_Copper_Cu, CR_CONDUCTOR_Gold_Au, CR_CONDUCTOR_Iron_Fe, CR_CONDUCTOR_Silver_Ag, CR_CONDUCTOR_ALL_CONDUCTORS }

  List of supported conductors (IOR and absorption), sampled at wavelengths 645nm, 526nm, 444nm. Source: https://refractiveindex.info.

- enum CRAY_TEXTURE_WRAPPING { CTW_CLAMP, CTW_REPEAT }

  List of supported texture wrapping modes.
- enum CRAY_TEXTURE_MAPPING { CTM_UV, CTM_PLANAR, CTM_SPHERICAL }

  List of supported texture mapping modes.
- enum CRAY_TEXTURE_FILTERING { CTF_BOX, CTF_TRIANGLE }

  List of supported texture filtering modes.

Functions

- void CR_init ()

  Initializes the rendering engine. This must be the first function call.
- void CR_destroy ()

  Destroys any internal allocations. This is only necessary if multiple rendering operations are to be performed.
- void CR_start ()

  Starts the rendering operation: performs scene preprocessing, initializes threads and starts rendering.
- void CR_stop ()

  Issues a request to stop any running rendering operations.
- void CR_clearScene ()

  Clears any internal scene data, e.g. cameras, acceleration structures, lights, materials, etc. This is needed only if a subsequent rendering operation is to be performed.
- bool CR_started ()

  Queries the renderer if a start command has been issued.
- bool CR_finished ()

  Queries the renderer if rendering has finished.
- void CR_enableRGBBuffer (bool enable)

  Enables/disables the use of an RGB buffer. By default, all results are written to a floating buffer, for increased precision. When an RGB buffer is enabled, the main writing is still performed on the floating buffer. The RGB buffer simply contains the converted 8-bit values after tonemapping and gamma correction. This is useful mainly for GUI operations, in order to avoid performing a whole conversion later on.
- void CR_setOutputName (const char *name)

  Sets an output name for writing. This is just a utility function, e.g. for scene loaders, as the engine does not perform any file operations. The name does not contain any extension. It is the responsibility of the caller to retrieve the final framebuffer and write to whichever file format is desired (or do anything with it).
- void CR_setFilmDimensions (int32_t width, int32_t height)

  Sets the image dimensions.
- void CR_setNumThreads (int32_t num_threads)

  Sets the number of threads to use for rendering. Ideally, this should be the number of cores in the system. The actual number of threads spawn is num_threads - 1 unless the use_main_thread flag is set to false. A value of 0 issues a serial operation.
- void CR_useMainThread (bool use_main_thread)

  Enables/disables the use of the main thread.
- void CR_setBlockSize (int32_t block_size)

  Sets the number of pixels per 2D tile to use for multithreaded rendering. A value of 16 or 32 is a typical use case.
- void CR_setSamplesPerPixel (int32_t spp)

  Sets the number of ray samples per pixel.
- void CR_setGamma (cr_float gamma)

  Sets the gamma correction value. This is used only when an RGB buffer is enabled.
- void CR_setExposure (cr_float exposure)

  Sets the exposure value to be used during the tonemapping operation. Internally, the traditional John Hable's Uncharted 2 tonemapping operator is used. This is used only when an RGB buffer is enabled. Using an exposure value of 0.0 degrades to no tonemapping. This process is then followed by gamma correction (see CR_setGamma)
- void CR_setAccelerationStructure (CRAY_ACCELERATION_STRUCTURE acceleration_structure_type)

Sets the acceleration structure. Simple scenes can use a simple array structure. For anything else, a BVH is the optimal option. Any subsequent calls to this function, overrides AND DELETES any previously allocated acceleration structure and internal data, e.g. primitives. This function should be called before adding any primitives.

- void CR_setIntegrator (CRAY_INTEGRATOR integrator_type)

  Adds an integrator. Currently, Ambient Occlusion and unidirectional Path Tracing are available. A scene can be rendered multiple times with different integrators, by simply issuing calls to this function and CR_start, i.e. no reloading of the scene data is required.
- bool CR_setInteractiveRender (bool enable)

  Enables/disables interactive rendering. This is useful only for integrations with a GUI. Enabling interactive rendering maintains all current settings with the following differences:
- void CR_useFastRNG (bool enable)

  Enforces the use of Xorshift RNG.
- bool CR_isRGBBufferEnabled ()

  Check if RGB buffer is enabled.
- const char * CR_getOutputName ()

  Gets the output name.
- float * CR_getFilmFloatDataPtr ()

  Gets a pointer to the internal float framebuffer. This should be called when rendering has finished, e.g. to write to a file.
- uint64_t CR_getFilmFloatByteSize ()

  Gets the number of bytes for the float buffer.
- uint8_t * CR_getFilmRGBDataPtr ()

  Gets a pointer to the internal RGB framebuffer. This should be called when rendering has finished, e.g. to write to a file.
- uint64_t CR_getFilmRGBByteSize ()

  Gets the number of bytes for the RGB buffer.
- int32_t CR_getFilmWidth ()

  Gets the width of the internal framebuffer.
- int32_t CR_getFilmHeight ()

  Gets the height of the internal framebuffer.
- int32_t CR_getFilmChannels ()

  Gets the number of channels of the internal framebuffer. This is currently fixed to 3.
- int32_t CR_getNumThreads ()

  Gets the number of threads.
- bool CR_usesMainThread ()

  Check if main thread rendering is enabled.
- int32_t CR_getBlockSize ()

  Gets the number of pixels per 2D tile.
- int32_t CR_getSamplesPerPixel ()

  Gets the number of ray samples per pixel.
- cr_float CR_getGamma ()

  Gets the gamma correction value.
- cr_float CR_getExposure ()

  Gets the exposure value used during the tonemapping operation.
- CRAY_ACCELERATION_STRUCTURE CR_getAccelerationStructure ()

  Gets the acceleration structure type used.
- CRAY_INTEGRATOR CR_getIntegrator ()

  Gets the integrator type used.
- bool CR_isInteractiveRendeEnabled ()

  Check if interactive rendering is enabled.
- bool CR_isFastRNGEnabled ()

  Check if Fast RNG is enabled.

- void CR_setAOSamplesPerPixel (int32_t spp)

  Sets the number of AO rays to spawn for each sample per pixel.
- void CR_setAORange (cr_float range)

  Sets the AO range of the rays. A value of 0 denotes infinite range.
- int32_t CR_getAOSamplesPerPixel ()

  Gets the number of AO rays spawned for each sample per pixel.
- cr_float CR_getAORange ()

  Gets the AO range of the rays.
- void CR_setRayDepth (int32_t depth)

  Sets the maximum depth of the rays for the path tracer. 1 is direct lighting, 2 is 1 indirect bounce, etc.
- void CR_setRussianRoulette (bool enable)

  Enables/disables russian roulette for path tracing. This function is also capped by the CR_setRayDepth.
- int32_t CR_getRayDepth ()

  Gets the maximum depth of the rays for the path tracer.
- bool CR_getRussianRoulette ()

  Gets the russian roulette status.
- cr_float CR_getDielectricIOR (CRAY_IOR_DIELECTRICS dielectric)

  Returns the internal index of refraction from a list of predefined dielectrics.
- CRAY_HANDLE CR_addMaterialDiffuseReflection (cr_vec3 diffuse_reflection_color)

  Creates a diffuse reflection material, modeled as a pure lambertian surface. It can be used to model matte materials.
- CRAY_HANDLE CR_addMaterialDiffuseTransmission (cr_vec3 diffuse_transmission_color)

  Creates a diffuse transmission material, modeled as a pure lambertian surface. It can be used to model translucent materials.
- CRAY_HANDLE CR_addMaterialDiffuse (cr_vec3 diffuse_reflection_color, cr_vec3 diffuse_transmission_color, cr_float eta_t)

  Creates a diffuse material representing rough reflection and transmission, modeled as a pure lambertian surface. It can be used to model translucent materials.
- cr_vec3 CR_getConductorIOR (CRAY_IOR_CONDUCTORS conductor)

  Returns the internal index of refraction from a list of predefined conductors.
- cr_vec3 CR_getConductorAbsorption (CRAY_IOR_CONDUCTORS conductor)

  Returns the absorption coefficients from a list of predefined conductors.
- CRAY_HANDLE CR_addMaterialConductor (cr_vec3 reflection_color, cr_vec3 absorption, cr_vec3 eta_t, cr_float roughness)

  Creates a conductor material.
- CRAY_HANDLE CR_addMaterialDielectricSpecularReflection (cr_vec3 reflection_color, cr_float roughness)

  Creates a dielectric material representing smooth/rough reflection with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model mirror materials.
- CRAY_HANDLE CR_addMaterialDielectricSpecular (cr_vec3 reflection_color, cr_vec3 transmission_color, cr_float eta_t, cr_float roughness)

  Creates a dielectric material representing smooth/rough reflection and transmission with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model glass materials.
- CRAY_HANDLE CR_addMaterialDielectricSpecularTransmission (cr_vec3 transmission_color, cr_float eta_t, cr_float roughness)

  Creates a dielectric material representing smooth/rough transmission with an isotropic Trowbridge-Reitz(GGX) microfacet model. It can be used to model purely transmissive surfaces.
- CRAY_HANDLE CR_addMaterialDiffuseSpecularReflection (cr_vec3 diffuse_reflection_color, cr_vec3 specular_reflection_color, cr_float eta_t, cr_float roughness)

  Creates a mixed dielectric material with both diffuse and specular (both smooth/rough) reflection components. The diffuse component is modeled as a pure lambertian surface, while the specular employs an isotropic Trowbridge-Reitz(GGX) microfacet model.
- CRAY_HANDLE CR_addImage (const uint8_t *ptr, int32_t width, int32_t height, int32_t channels, int32_t size_type, bool is_srgb)

  Creates an image. The data is internally allocated.

- CRAY_HANDLE CR_addImageCheckerboard (cr_vec3 tex1_rgb, cr_vec3 tex2_rgb, cr_float frequency)

  Creates a checkerboard texture using two different colors.
- CRAY_HANDLE CR_addTextureSampler (CRAY_TEXTURE_MAPPING mapping_mode, CRAY_TEXTURE_WRAPPING wrap_mode, CRAY_TEXTURE_FILTERING filtering_mode)

  Creates a sampler.
- CRAY_HANDLE CR_addTexture (CRAY_HANDLE image_id, CRAY_HANDLE sampler_id)

  Associates a texture with a sampler to be used for rendering.
- void CR_addMaterialDiffuseReflectionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's diffuse reflection color.
- void CR_addMaterialDiffuseTransmissionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's diffuse transmission color.
- void CR_addMaterialSpecularReflectionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture_id)

  Associates a texture with a material's specular reflection color.
- void CR_addMaterialSpecularTransmissionTexture (CRAY_HANDLE material_id, CRAY_HANDLE texture↩_id)

  Associates a texture with a material's specular transmission color.
- void CR_pushMatrix (cr_mat4 matrix)

  Pushes a transformation matrix onto the stack.
- void CR_popMatrix ()

  Pops a transformation matrix from the stack.
- cr_mat4 CR_rotate (cr_float degrees, cr_vec3 axis)

  Utility function to generate a rotation matrix along an arbitrary axis.
- cr_mat4 CR_translate (cr_vec3 translate)

  Utility function to generate a translation matrix.
- cr_mat4 CR_scale (cr_vec3 scale)

  Utility function to generate a scale matrix.
- cr_mat4 CR_mulMatrix (cr_mat4 matrix1, cr_mat4 matrix2)

  Utility function to multiply two matrices, M = M1 ∗ M2.
- void CR_addCameraThinLens (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float aperture, cr_float far, cr_float lens_radius, cr_float focal_distance)

  Creates a thinlens camera. This is essentially the perspective camera with a predefined lens radius and focal distance.
- void CR_addCameraPerspective (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float aperture, cr_float near, cr_float far)

  Creates a perspective pinhole camera. This is essentially the thinlens camera without a predefined lens radius and focal distance.
- void CR_addCameraOrthographic (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float height, cr_float near, cr_float far)

  Creates an orthographic camera.
- void CR_addLightPointSpot (cr_vec3 position, cr_vec3 target, cr_vec3 intensity, cr_float falloffAngle, cr_float maxAngle, cr_float exponent)

  Creates a point spotlight. This is a dirac light.
- void CR_addLightPointOmni (cr_vec3 position, cr_vec3 intensity)

  Creates a point omnidirectional light. This is a dirac light.
- void CR_addLightDirectional (cr_vec3 direction, cr_vec3 intensity_rgb)

  Creates a parallel light using a direction.
- void CR_addLightDirectionalFromTo (cr_vec3 position, cr_vec3 target, cr_vec3 intensity_rgb)

  Creates a parallel light using a point and a target to generate a direction.
- CRAY_HANDLE CR_addLightAreaDiffuseEmitter (cr_vec3 flux, bool doublesided)

  Creates an area light, modeled as a diffuse emitter. This needs to be associated with a primitive. Currently, triangles are not yet supported.
- void CR_addAreaLightToPrimitive (CRAY_HANDLE primitive_id, CRAY_HANDLE light_id)

Associates an area light with a primitive. Note that the same light source can be shared by multiple primitives. In that case, the light's flux is scaled by each primitive's area. This way, it is easy to associate a large polygon group with one light source. Since this method is approximate, accurate rendering requires 1 light per primitive.

- void CR_reservePrimitives (size_t size)

  Reserves memory for the requested number of primitives.
- void CR_reserveMaterials (size_t size)

  Reserves memory for the requested number of materials.
- size_t CR_getNumPrimitives ()

  Retrieves the current number of primitives.
- size_t CR_getNumMaterials ()

  Retrieves the current number of materials.
- CRAY_HANDLE CR_addPrimitiveSphere (cr_vec3 position, cr_float radius, bool flip_normals, bool double↩ _sided, CRAY_HANDLE material_id)

  Creates a sphere primitive.
- CRAY_HANDLE CR_addPrimitiveRectangleXZ (cr_float min_x, cr_float max_x, cr_float min_z, cr_float max_z, cr_float offset, bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an axis-aligned rectangle primitive on the XZ axis.
- CRAY_HANDLE CR_addPrimitiveRectangleXY (cr_float min_x, cr_float max_x, cr_float min_y, cr_float max_y, cr_float offset, bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an axis-aligned rectangle primitive on the XY axis.
- CRAY_HANDLE CR_addPrimitiveRectangleYZ (cr_float min_y, cr_float max_y, cr_float min_z, cr_float max_z, cr_float offset, bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an axis-aligned rectangle primitive on the YZ axis.
- CRAY_HANDLE CR_addPrimitiveTriangle (bool flip_normals, bool double_sided, CRAY_HANDLE material_id)

  Creates an empty triangle primitive.
- void CR_addPrimitiveTriangleVertexPositions (CRAY_HANDLE triangle_id, int32_t index, cr_vec3 position)

  Associates a group of vertices with an existing triangle.
- void CR_addPrimitiveTriangleVertexNormals (CRAY_HANDLE triangle_id, int32_t index, cr_vec3 normal)

  Associates a group of normals with an existing triangle. If these are not provided, triangle plane normals are used.
- void CR_addPrimitiveTriangleVertexTexcoords (CRAY_HANDLE triangle_id, int32_t index, cr_vec3 texcoord)

  Associates a group of texcoord with an existing triangle.
- void CR_setMinimumLogLevel (CRAY_LOGGERENTRY type)

  Sets the minimum log level to store messages.
- CRAY_LOGGERENTRY CR_getMinimumLogLevel ()

  Gets the minimum log level.
- void CR_setLogFile (void *file)

  Enables writing to an output FILE. If disabled, the messages are written to a queue and can be polled using CR_getLastLogMessage.
- void * CR_getLogFile ()

  Gets the current output FILE.
- void CR_printTostdout (bool enable)

  Enables writing internal log messages. If disabled, the messages can be polled using CR_getLastLogMessage.
- bool CR_isPrintTostdoutEnabled ()

  Returns the status of printing to stdout.
- void CR_printProgressBar (bool enable)

  Enables of progress bar to stdout. If disabled, the progress percentage can be polled using CR_getProgressPercentage. Note that stdout printing must be enabled (see CR_printTostdout).
- bool CR_IsPrintProgressBarEnabled ()

  Returns the status of printing the progress bar to stdout. It is suggested to disable the progress bar when verbose logging is enabled, as it may interfere with the progress bar printing.
- cr_float CR_getProgressPercentage ()

Retrieves the progress percentage of a running rendering process.

- bool CR_getLastLogMessage (char *msg, size_t *msg_length, CRAY_LOGGERENTRY *type)

  Gets the last log message stored by the rendering engine.

- bool CR_debug_intersectPixel (cr_vec3 *position, int32_t pixel_x, int32_t pixel_y)

  Debug utility that returns the intersection position at a particular pixel.

- void CR_debug_updateCameraPerspective (cr_vec3 position, cr_vec3 target, cr_vec3 up, cr_float aperture, cr_float near, cr_float far)

  Debug utility that updates camera parameters for navigation during the interactive mode. This is only useful as part of a GUI.

- void CR_debug_setTestPixelRange (int32_t start_x, int32_t start_y, int32_t end_x, int32_t end_y)

  Debug utility that logs verbose messages in debug mode for a particular pixel range [start, stop]. Very useful for cases where internal values need to be viewed. Enable/Disable with CR_debug_setTestPixelRange.

- void CR_debug_enableTestPixelDebug (bool enable)

  Debug utility to enable logging of verbose messages in debug mode for a particular pixel range [start, stop].

- bool CR_debug_isTestPixelDebugEnabled ()

  Debug utility that returns if debug pixel printing is enabled.

Index