Contents

Part I

Manual

Chapter 1

Introduction

1.1 What is Beast?

Beast is an offline Global Illumination rendering solution targeted towards baking. It is designed to be integrated in the editor or the content pipeline of a game or other program to precompute lighting. The intention is to be able to bake any kind of lighting to any basis on any surface. We try to provide a tool that can produce data for your lighting pipeline rather than enforcing a lighting pipeline on you.

1.2 What is the Beast API?

The Beast API is Illuminate Labs way to give users of Beast an easy and consistent way to integrate offline rendering technology into game engines and modeling packages. It hides a lot of the work that previously had to be done by the integrator. This makes it easier to get started and simplifies building integrations using more of Beasts features. It also makes it possible for us to change how Beast works internally without affecting existing integrations.

1.2.1 Design Goals

The Beast API should be:

- Easy to use
  The main goal with Beast API is to make it as easy as possible to build a Beast integration for any application.
- Concise and Consistent
  The interface to the Beast API works in the same way in different places as much as possible. There is also an effort to provide only one way of doing each task in order to provide a consistent experience to the API user.
- Non-intrusive and Compatible
  The library is built not to clash with external code or libraries. It enables clients to

be written in many different programming languages and environments (for example C++, Python and .NET).

- Data driven
  The API is focused on what to do, not how to do it. It should make no difference for the integrator if the rendering is done on the same computer or a cluster.

1.3 Current State of the Beast API

This release of the Beast API supports the complete workflow from scene definition, render passes, target specification and as well as getting the results of the job back through the API. The next version of the API will additionally include support for controlling Global Illumination settings through the API, which currently is done in an XML file as described in the Beast manual.
This and more can be found on the Beast site
http://www.illuminatelabs.com/beast.

1.4 Feedback

We are always interested in feedback on how the API works in real life. Let us know if you find problems or have any suggestions! You can reach us on support@illuminatelabs.com.

Chapter 2

System Overview

2.1 Deployment

Beast is a standalone renderer available as an external executable. The Beast API helps out with scene export and execution of the external application through a programming API. What happens behind the scene is that the API calls writes data such as textures, meshes and scene definitions to disk and invokes thebeast.exe on the content. This means that the rendering will be done in a separate process with its own address space. The deployment model based on writing to disk and invoking thebeast.exe may change but the intention is that the API will stay the same.

2.2 Setting up Scenes

To get Beast to generate lighting you need to mirror your own scenes or levels in Beasts representation. This is done using triangle meshes, light sources, cameras and materials. What follows here is a description of each component in a scene.

2.2.1 Meshes

Objects are represented as triangle meshes in Beast. When working with other representation you need to convert them before feeding them into Beast.

Render Stats

Render Stats override how objects behave for different kinds of situations. This can be used to disable shadow casting on an object or exclude it from influencing the global illumination solution etc.

2.2.2 Materials

The material model in Beast is based on the classic Phong shader with textured or vertex colored channels.
This might seem limiting but a light map or light probe stores incoming light which means it only shows the effects of the material indirectly, i.e through light bounced off the material. You can map the following channels:

- Diffuse
- Specular
- Emissive
  Also known as incandescence. Light being emitted from the surface.
- Transparency

Additionally, there are values set per material:

- Reflectivity
- Shininess
  Also known as Phong exponent. How specular a material is. A high value gives a sharp highlight, a low one gives a fuzzy spread out highlight.

2.2.3 Light Sources

The Beast API supports many different light sources. All have different attributes to model their intensity, color and shadow casting properties.

Point Light

Also called omni lights. Emits light equally in all directions. It is possible to configure its falloff and shadow casting properties.

Spot Light

Technically a point light that has all but a cone occluded. It is possible to configure its falloff, shadow casting properties and the size of the penumbra. It's also possible to project textures from it.

Directional Light

Also called sun light. A distant directional light without any falloff. It is possible to configure its angle coverage to give soft shadows.

Area Light

Rectangular light emitting surfaces. Physically accurate and powerful primitives to light levels with. It is possible to configure the size of it and the shadow casting properties.

Window Light

Rectangular light emitting "windows". Works as a directional light passing through a window. The direction of the the sun is the same as the normal of the window light. It is possible configure the size of the window and the angle of the sky the sun covers for soft shadows.

Sky Light

Beast supports using skylights based on constant colors and HDR images. This is currently not exposed in the API, the section on configuration XML describes how to use it. Sky Lights will be exposed through the API in the future.

Objects as light sources

When global illumination is enabled objects with emissive materials will act as light sources. Generally large and dim light sources can be modeled as emissive materials. Small intense lights should preferably be modeled as light sources or they might introduce noise in the renderings.
The section about the configuration XML describes how to enable global illumination.

2.2.4 Cameras

Beast currently supports two camera types. Perspective projection cameras, and environment map cameras.

Perspective Cameras

Perspective cameras are defined by their Field of view and a pixel aspect ratio.

Environment Cameras

Environment cameras renders environment maps. They support the following projections

- Cube Map
- Environment Sphere Map
- Lat Long Map

2.3 Render Passes

Render passes control what gets stored on the surface of baked textures or in the vertices when vertex baking. The render pass is added to a render target. One texture will be generated for each render pass on each render target.

2.3.1 Types

These are the render passes currently supported in the Beast API. Please note that there are some restrictions on some pass/target combinations. The term fragment is used in the pass descriptions to refer to the texel, vertex, pixel or point cloud point that is currently being rendered or baked.

Full Shading Pass

The result of the full shading pass is the fully shaded fragment, including direct light, indirect light and the material color of the fragment.

Illumination Pass

The illumination pass outputs only the incoming light at the fragment. The user can select whether to output direct light, indirect light or both.

RNM Pass

The RNM pass (RNM is short for Radiosity Normal Map, also called Directional Light Map) calculates the light for a number of normal direction which deviate from the normal of the fragment currently shaded. These deviations are called the RNM basis. The result from each normal direction is stored separately. The RNM pass can like the illumination pass store direct light, indirect light or the full light contribution.

Light Pass

The light pass is used to bake maps including only certain light sources. This could be used to create static shadow map textures for light sources in order to be able to change the color/intensity of the light source in the shader.

Ambient Occlusion Pass

Ambient occlusion is the visibility function integrated over the hemisphere of the fragment. This pass outputs ambient occlusion.

LUA Pass

The LUA pass takes a LUA script as input for controlling the baking. This enables among other things storing light in custom bases. For more information about this, please see the Beast Technical Reference.

2.4 Render Targets

Render targets specifies what will be rendered or baked. The render pass belongs to a render job. To actually output something one or more render passes is added to the target. When the job is executed all its targets will yield one output for each render pass added to each target.

2.4.1 Types

These are the render targets available in the Beast API.

Camera Target

Renders the scene from the point of view of a camera. The output is an image in the desired resolution.

Texture Target

The texture target will perform standard texture baking. The output is a single texture. All instances added to the target will be baked together in one UV space.

Atlased Texture Target

The atlased texture target will perform atlasing on the instances added to the target. This means that each instance will be placed in an unoccupied part of UV space and more than one texture will be baked if necessary. After the baking the position of each instance can be retrieved.

Vertex Target

Performs baking of each vertex on all instances added to the target. The output is the baked data for each vertex.

Pointcloud Target

Works like Vertex Target but instead of adding instances with shapes the input is arbitrary points and normals instead. The output is the baked data for each point in the cloud.

2.5 Render Jobs

The Render Job is a model for controlling the invocation of Beast from the Beast API.

2.5.1 Input

The input of a render job is a scene object and an XML file defining the render settings for the job. Render targets are then created from the job object and automatically associated with the job.

2.5.2 Running and monitoring progress

When all targets and passes have been added to the job it can be started. Running the job will execute all targets and their respective render passes. While it is running the client application can monitor the progress at any time. It can either check the status of the job at regular intervals or wait until there is new information available. If there are any errors running Beast the job will be aborted and the client can get information about the error.

2.5.3 Output

When the job has finished successfully the output is available through the API. For texture, atlas and camera targets the results are images. For vertex and point cloud targets the results are points. For atlas targets the position and scaling of each entity in the atlas is available as well. Each pair of render pass and target that has been added to the job for rendering will when the job has finished have a framebuffer or set of points which can then be retrieved.

2.6 Configuration files

The configuration XML file defines how the rendering is done. It contains the following information:

- Global Illumination
  What GI algorithms to use, what quality settings to use.
- Other Quality Setting
  Super sampling, Ray Depths, etc.
- Environment Settings
  Environment Images, Skylight colors for different kinds of ray types.

There is a complete reference on the config XML file in the Beast Manual. The Integrators Kit has examples on how to setup XML files as well.

Chapter 3

Programming Manual

3.1 Overview

Beast is delivered as a dynamic library containing only C symbols for maximum compatibility. Almost any programming language on the Windows platform can load DLL files and import C functions. The DLL is available in both 32 bit and 64 bit library for Windows.
C is not the most powerful language around and the intention is that calls to Beast should be wrapped up in the environment that is used to avoid problems with resource leaks, manual memory management etc. All Beast types, functions and constants are prefixed ILB to avoid namespace conflicts. If any problems of that kind happens, let us know!

3.1.1 Examples

Beast comes with samples. This is a good way to see how the API functions work in a context.

3.1.2 Beast API and Development Environments

To get Beast up running as a developer, the directory `beastapi\include` has to be added to the include directories. The `beastapi\lib` has to be added to the library directories. On a 32 bit system, add `beast32.lib` to the list of used libraries, 64 bit users should add `beast64.lib`. When this is set up it should be possible to include Beast API headers like this:

```
    #include "beastapi/beastmanager.h"
```

3.1.3 The DLL

When working with a dynamic library, in our case beast32.dll and beast64.dll, it needs to be placed in a location where the program using it can find it. These are the options:

- Put it in the same directory as the executable
- Put it in the PATH
  A program looks for DLL files in directories in the environment variable PATH.
- Install the DLL Globally
  A DLL placed in the windows\system32 directory can be found by all programs. This also has problems since multiple versions of the DLL cannot coexist.

Another approach is to load the DLL manually using **LoadLibrary**. This needs some extra effort. Refer to integration help for more information about this.

3.1.4 The Beast Executable

The Beast API uses the binary thebeast.exe (or thebeast-64.exe on 64 bit machines) for rendering. It looks for it in the following paths (in the specified order):

- The directory specified in the environment variable %BEAST_ROOT%\bin.
- The same directory as the Beast DLL is loaded from.

It can also be explictly set using the API call ILBSetBeastPath. After a call to this function it will stop searching the other paths.

3.1.5 The Beast License

Beast searches several places in order to find a valid license. The client needs to supply a valid license in at least one of these places:

- The same directory as the Beast binary
- The file pointed to by the environment variable IL_LICENSE_FILE
- The directory pointed to by the environment variable IL_LICENSE_DIR
- In the file C:\illm\tb.dat

3.1.6 Handles

All non trivial types such as scenes, textures and render jobs are defined as handles. With each handle type there are a set of functions to operate on it with. Together these map closely to classes in object oriented languages. Think of the handle as the *this* pointer and the functions as methods. The typical way to work is by first aquiring a handle using a factory function and then start operating on it:

```
ILBBeastMesh myMesh;
// Note how myMesh is passed as a pointer
// that is being initialized by ILBBeginMesh
ILBBeginMesh(bm, "myMesh", &myMesh);

// Now myMesh is passed as value
ILBAddVertexData(myMesh, vertexData, normalData, vertexCount);
...
```

3.1.7 Strings

Beast uses strings in many situations, typically for resource names and messages.

String Encodings

Beast handles input and output of strings using the following encodings

- ANSI The default encoding in visual studio when working with 8 bit strings. This is what to use if the application does not need Unicode support. Strings work like char* with ASCII characters only.

- UTF-8 Unicode 8 bit encoding.
- UTF-16 Unicode 16 bit encoding, default in Visual Studio when working with Unicode strings.

The Beast API will try to auto detect what string encoding the program uses with the preprocessor, but if the check fails it can be overridden using one of the following defines:

- #define ILB_STRING_UTF8
- #define ILB_STRING_UTF16
- #define ILB_STRING_ANSI

When overriding the string encoding, make sure it is overridden in all files directly or indirectly including Beast headers, otherwise there will be problems. The auto detection and setup for strings are in the file beastapitypes.h.

Input Strings

When inputting strings to Beast the type ILBConstString is used. It is a pointer to a zero terminated string of characters. The character type is different depending on what string encoding is used. This ensures that it can take string constants based on char* if ANSI or UTF-8 is used and string constants based on wchar_t* when UTF-16 is used.

Output Strings

Output strings requires memory management which means that they need some extra effort. The type used for this purpose is ILBStringHandle. The interface is to first query the length of the string and then allocate a buffer of correct size and copy it back. It will use the same encoding as configured for input strings. The functions will give the length of the string in number of ILBCharType which depends on what string encoding is being used. The length will include the 0 termination marker. It will never return 0 characters of data. Empty strings are represented as a single 0 termination marker. The API documentation for beaststring.h contains details on each function.
The typical way of handling return strings is to create a function that converts the string to the format used in the host application. Here is example code (without error handling) for creating a std::string from a ILBStringHandle in C++:

```
std::string convertBeastString(ILBStringHandle h) {
    int32 len;
    ILBGetLength(h, &len);
    // Note that Beast strings says its length
    // including zero termination, we need to
    // make the std::string 1 character shorter
    // since it expects sizes without zero
    // termination
    std::string result(len - 1, 0);
    ILBCopy(h, &result[0], len);
```

```
        ILBReleaseString(h);
        return result;
    }
```

3.1.8 Error Handling

All Beast API functions returns a ILBStatus which is an enum defined in beastapitypes.h. If getting a ILB_ST_SUCCESS as a result of a call it succeeded. The other status codes says something about the nature of what happened. Note that a status different from ILB_ST_SUCCESS is not necessarily an error. An example of this is ILB_ST_UNKNOWN_OBJECT which you can get when you check if an object is cached, which is a completely normal case.

Extended Error Information

An error code does not always give a complete idea on what went wrong. Use the function ILBGetExtendedErrorInformation to get more information about what happened. The function is thread specific so if an error code is given in one thread ILBGetExtendedErrorInformation must be called from the same thread to get the extra information. It will only return the last error you got.

Error Handling in C++

Many developers prefer to use exceptions rather than return codes for error handling to get cleaner code. A simple way of mapping Beast errors to exceptions is to introduce a help function that captures the return value and throws it as an exception:

```
    class BeastException {
    public:
        BeastException(ILBStatus s) : status(s) {}
        ILBStatus status;
    };
    inline void beastCall(ILBStatus s) {
        if(s != ILB_ST_SUCCESS) {
            throw BeastException(s);
        }
    }
```

Now you can use beastCall to generate exceptions for you and handle them using a try / catch block:

```
    try {
        ...
        beastCall(ILBSetFov(cam, PI / 3.0f, .1f));
        ...
    } catch(BeastException& ex) {
        ILBStringHandle errorString;
        ILBStringHandle extendedError;
        ILBErrorToString(ex.status, &errorString);
        ILBGetExtendErrorInformation(&extendedError);
        std::cout << "Error:" << convertBeastString(errorString) << std::endl;
        std::cout << "Info:" << convertBeastString(extendedError) << std::endl;
    }
```

Crashes and Unhandled Exception

If you get an error of the type ILB_ST_UNHANDLED_EXCEPTION make sure you contact Illuminate Labs support on support@illuminatelabs.com about it. This kind of errors should not happen and we will do our best to solve them quickly. Crashes should generally not happen, but using invalid or uninitialized handles could cause this so make sure this is not the case before sending crash reports.

3.1.9 Logging

Beast provides logging of errors and information. The destination of the log messages is set using ILBSetLogTarget. The error and information logs can be directed to different streams by calling the function multiple times. The supported target streams are:

- ILB_LS_NULL
  The messages are discarded (default)
- ILB_LS_STDOUT
  The messages are printed to stdout
- ILB_LS_STDERR
  The messages are printed to stderr
- ILB_LS_FILE
  The messages are printed to a user specified file
- ILB_LS_DEBUG_OUTPUT
  The messages are printed to the output console in visual studio if a debugger is connected.

Note that calls to these functions should be done before creating any BeastManagers and that this setting is global rather than BeastManager specific since logging of information sometimes needs to be done before a Beast Manager is initialized.

3.1.10 Thread Safety

Generally Beast API calls can be done from many threads simultaneously without problems. The exceptions is all form of Destroy/Release calls which can only be called once and must not be called while another thread is inside a call on the object. Also when creating the different kinds of primitives (i.e Meshes, Instances, etc.) only one thread can make calls on the same primitive at the same time. It is no problem creating two primitives in parallel from two different threads though. Every type has its own section called Thread Safety that goes through any special cases.

3.2 Vectors, Matrices and Colors

Vectors, matrices and colors are defined as C structures in beastapitypes.h. There are #ifdef:ed constructors in them enabled for C++ developers. The structs are not intended as high level math primitives, but rather as data carrying structs. It is possible to inherit and extend them in C++, but make sure not to introduce any virtual methods since it would change their size and that will cause problems when passing them to Beast. Please refer to Appendix A for more information on vectors and coordinate systems in Beast. Appendix B contains information about colors. The complete list of vector and color classes are:

- ILBVec2
  Two dimensional vectors. Used for UV coordinates.
- ILBVec3
  Three dimensional vectors. Used for Points and directions. The interpretation should be obvious from the context.
- ILBLinearRGB
  RGB Color in linear color space.
- ILBLinearRGBA
  RGB Color with alpha in linear color space.
- ILBMatrix4x4
  A 4x4 Matrix. Implemented as a struct with a single member m of 16 floats. The correct way of addressing it is [row * 4 + column].

3.3 Beast Manager

The Beast Manager is the central object in all Beast usage. All resources are directly or indirectly managed through the Beast Manager. It works as a factory for most other types. It also manages the disk caching of resources. When destroying a Beast Manager all handle types that was created directly or indirectly through it will be invalid. It is possible to have multiple active Beast Managers running in the same program, they need to have separate cache directories though.

3.3.1 Caching

When creating a Beast Manager a cache directory and a cache scope is provided. The directory is a path to a directory that Beast is allowed to write to. It is also important that it can delete files in it.
The cache scope specifies if the cache is local to this instance of the Beast manager or if it is global. If running in global mode, you can keep the cache between different runs and a mesh or texture you cached previously will still be present the next time you initiate a globally scoped Beast Manager in that directory. This also means it is important that if you change a resource, you either invalidate it in the cache or make sure it gets a new name. A good policy to get this running is to embed a unique id or a revision number in all cached resource names. Make sure you update revision or UID every time you update the resource. This will make sure names will not conflict with previous versions. If you load an older scene with an old version of the resource it will not use the updated version in the cache since it has a different name. You can manually erase cached textures or meshes using the methods ILBEraseCachedMesh and ILBEraseCachedTexture respectively.
When working with cached resources, you should always be ready to create it again. When acquiring a handle you should check in the cache using ILBFindMesh or ILBFindTexture. If it is not in the cache you will have to create it again. You can clear the entire cache using ILBClearCache

3.3.2 Thread Safety

All calls to the Beast Manager except ILBDestroyBeastManager are thread safe. This means that the Beast Manager cannot be destroyed if other threads are working on it.

3.3.3 Declaration

The Beast Manager functions are found in the file beastmanager.h

3.4 Meshes

In the Beast API all meshes are specified as triangle meshes using indexed triangle lists. All animation and tessellation has to be done by the integrator. A mesh is not an instance, meaning that by just creating a mesh it does not automatically get a presence in the world. To actually place it, create an instance with a transform that locates it physically in the world.
The mandatory parts of creating a mesh is adding vertices, normals and triangles. You can also optionally add UV sets, vertex colors, tangents and bitangents. Some passes, most notably the RNM pass requires tangents and bitangents.
The syntax for creating a mesh is:

```
// Create the mesh
ILBBeastMesh mesh;
ILBBeginMesh(bm, "MyMesh", &mesh);

// Add one or many batches of vertices / normals
ILBAddVertexData(mesh, points, normals, pointCount);

// First group of triangles sharing the same material
ILBBeginMaterialGroup(mesh, "material1");
ILBAddTriangleData(mesh, index, indexCount);
ILBEndMaterialGroup(mesh);

// Another group of triangles sharing a different material
ILBBeginMaterialGroup(mesh, "material2");
ILBAddTriangleData(mesh, index2, index2Count);
ILBEndMaterialGroup(mesh);

// Add a UV layer.
ILBBeginUVLayer(mesh, "uv1");
ILBAddUVData(mesh, uv1, uv1Count)
ILBEndUVLayer(mesh);

// Add a color layer
ILBBeginColorLayer(mesh, "color1");
ILBAddColorData(mesh, colors, colorCount);
ILBEndColorLayer(mesh);

// Add tangents and bitangents
ILBBeginTangents(mesh);
ILBAddTangentData(mesh, &tangents[0], &bitangents[0], vertexCount);
ILBEndTangents(mesh);

// End the mesh
ILBEndMesh(mesh);
```

Polygons, UV coordinates, tangents and colors can be added in any order but each state has to be ended before going to the next.
All adding of data, i.e calls to ILBAddVertexData, ILBAddTriangleData, ILBAddUVData, ILBAddColorData and ILBAddTangentData can be broken up in many batches to save temporary memory. This means it is possible to balance memory overhead against function calls if source mesh data structures does not match the ones of Beast.
After calling ILBEndMesh, the mesh cannot change. To modify it, it has to be removed from

the cache and recreated.

3.4.1 Triangles and Material Groups

All triangles are created in a material group that shares the same material. When creating a Material group on a mesh the material is specified by name as opposed to a handle since the mesh is a cached resource that can be used in many scenes but the material is local to the scene. This means that the Beast API cannot check if the material exist at the time the mesh is created. When creating an instance of the mesh it will lookup the material with the same name in the scene it is being used in. For a more safe way of using materials consider using the per instance material instead.

3.4.2 Caching

Meshes are cached objects in the Beast API, this means that they can be used in many scenes at the same time. This also means care has to be taken about names and revisions when using meshes. Please refer to the section about caching for more information about this.

3.4.3 Thread Safety

The calls to ILBCreateMesh, ILBFindMesh etc. are mutexed so you can create and lookup new meshes from many threads. However only one thread may call a changing function on the same mesh at the same time. This means that you can have multiple threads creating meshes, but each single mesh should be created in only one thread.

3.4.4 Declaration

The Beast Mesh functions are found in the file beastmesh.h

3.5 Textures

There are two ways of specifying textures, either by referencing in textures from disk or by specifying them as pixel data through the API.

3.5.1 Referencing Texture Files

Beast can load textures in the following formats.

- TGA
- EXR
- PNG
- BMP
- DDS
- JPG
- TIFF
- HDR

To reference a texture use the function ILBReferenceTexture It takes both a unique name specifying what it should be called for caching purposes and a filename that locates the file physically on disk. The unique name can be the same as the filename, but can also be completely unrelated. Referenced textures are being copied to the cache directory rather than used directly from where it is located.

3.5.2 Creating Textures from Pixel Data

Beast supports Low Dynamic Range (LDR) and High Dynamic Range (HDR) textures. The textures can be monochrome, RGB (color) or RGBA (color and alpha channel). Creating a texture is done by adding pixels laid out line by line ordered from bottom to top.

3.5.3 Gamma Correction

For users not familiar with gamma correction, please refer to the section about color spaces. HDR textures are specified in linear space. It is the integrators responsibility to convert the pixel data to linear data.
When creating LDR textures it is possible to specify the input gamma. The gamma is given as the gamma for the screen the image is created on. In environments where gamma is ignored, the right setting is most likely 2.2 which is also the default. Otherwise it can be set using the function ILBSetInputGamma.

3.5.4 Examples

The typical workflow to create a texture looks like this:

```
ILBTextureHandle tex;
// Create a texture
ILBBeginTexture(bm, "texture1", width, height, ILB_PF_RGBA_BYTE, &tex)
// Add pixeldata
ILBAddPixelDataLDR(tex, pixels, width * height);
// Done, finalize it!
ILBEndTexture(tex)
```

Note that ILBAddPixelDataHDR and ILBAddPixelDataLDR can be called multiple times to avoid having to create a cloned version of the entire image if using a different line order or color component order than Beast.

```
ILBTextureHandle tex;
// Create a texture
ILBBeginTexture(bm, "texture2", width, height, ILB_PF_RGBA_BYTE, &tex)
    // Add data line by line
    for(int y = 0; y < height; ++y) {
        // Prepare a scanline of data
        char* lineData = prepareLine(y);
        // Add a scanline of data
        ILBAddPixelDataLDR(tex, lineData, width);
    }
// Done, finalize it!
ILBEndTexture(tex)
```

3.5.5 Caching

Textures are cached objects in the Beast API, this means that they can be used in many scenes at the same time. This also means you will have to be careful about names and revisions when using textures. Please refer to the section about caching for more information about this.

3.5.6 Thread Safety

The calls to ILBCreateTexture, ILBFindTexture etc. are thread safe so it is possible to create and lookup textures from many threads. However only one thread may add pixel data to a specific texture at the same time.

3.5.7 Declaration

The Texture functions are found in the file beasttexture.h

3.6 Scenes

A scene is what turns meshes and textures into something renderable. It consists of:

- Cameras
- Object Instances
- Light Sources
- Materials
- Point Clouds

The scene object works as factory for all these object types and their memory is managed by the scene they belong to. Any handle to any of these objects will be invalidated after you have released the scene it was created by.
There is no concept of a scene graph or hierarchy in scenes, Beast expect the integrator to flatten all transformations you use to place your scene objects with.
Scenes are considered light weight objects which means they are not cached. They are also static in the sense that when you have called ILBEndScene you cannot modify it.

3.6.1 Thread Safety

It is possible to create scene objects from many threads in parallel. ILBEndScene Should be called after all creation calls are finished. ILBReleaseScene should be called only once while no other calls on the scene are active.

3.6.2 Declaration

The Scene functions are found in the file beastscene.h

3.7 Instances

Instances places meshes in the world. An instance is in its essence a transform from object to world space. They also allow setting and overriding materials, render stats and light links.

3.7.1 Render stats

Render stats are settings on objects that alter their behavior in the world for different situations. The complete list of available render stats can be found in the API reference for ILBRenderStats. To alter render stats, use the function ILBSetRenderStats. A typical use of the function can look like this:

```
// Make the object cast no shadows and receive no shadows
// Note how we combine multiple render stats using |
ILBSetRenderStats(targetInstance,
                  ILB_RS_CAST_SHADOWS | ILB_RS_RECEIVE_SHADOWS,
                  ILB_RSOP_DISABLE);
```

3.7.2 Light Links

Light links are a way of making an instance only be affected by some of the lights in the scene. For a complete overview of light links, please refer to the Light Links section.

3.7.3 Material Overrides

Materials of a mesh can be overridden on the instance level. A list of materials is given as input to the ILBSetMaterialOverrides function. Currently only one material override for the entire instance is supported, rather than one per material group. The workaround for this is to export a separate mesh for objects with different material group setups.

3.7.4 Thread Safety

You should only operate on the same instance from one thread at the time.

3.7.5 Declaration

The Instance functions are found in the file beastinstance.h

3.8 Light Sources

Beast has numerous light sources. To create lights, use the ILBCreate*Light family of functions found in beastlightsource.h. The transform matrix parameter is used to place and orient light sources. The position and direction of a light source is given by a matrix with a translation and rotation from the default negative Y direction. Scaling in the matrix controls the size of area lights and window lights as well as the scale of the light source ramp if enabled. The intensity parameter is a colored intensity for how strong the light source is. Many light sources have extra properties such as cone angles, shadow radius etc.

3.8.1 Shared Light Properties

All lights can use the following functions to control their behavior:
ILBSetCastShadows. Enables or disables shadow casting. Disabled by default.
ILBSetShadowSamples. Sets the number of shadow rays to shoot towards the light to give soft shadows. Exactly how to setup the shadow characteristics for the light depends on its type. Set to 1 by default.
ILBSetIntensityScale. Sets the scale for direct and indirect light intensity. A typical use for this is to disable direct light for a light source. This makes it possible to add direct lighting with high resolution shadows inside the game, but let the indirect light be baked into a light map. Direct and indirect scale is 1 by default.

3.8.2 Light Types

Point Light

The point light is created using ILBCreatePointLight. It is possible to control its behavior using the functions:
ILBSetShadowRadius. Sets the world space radius of the light source for shadow casting purposes. Default is 0
ILBSetFalloff. Sets the falloff function for the light source. It supports two kinds of falloff ILB_FO_EXPONENT and ILB_FO_MAX_RANGE. It is set to no falloff by default.
ILBSetLightRampEntry. Sets a ramp of colors as falloff for the light source. Can be used approximating custom falloff settings. Disabled by default.

Spot Light

The spot light is created using ILBCreateSpotLight It is possible to control its behavior using the functions:
ILBSetShadowRadius. Sets the world space radius of the light source for shadow casting purposes. Default is 0
ILBSetFalloff. Sets the falloff function for the light source. It supports two kinds of falloff ILB_FO_EXPONENT and ILB_FO_MAX_RANGE. It is set to no falloff by default.
ILBSetLightRampEntry. Sets a ramp of colors as falloff for the light source. Can be used approximating custom falloff settings. Disabled by default.

ILBSetSpotlightCone Customizes the spotlight cone. Controls the size of the cone, the penumbra angle (how large the "fade out" region of the cone is) and an exponent to control the shape of the penumbra gradient.
ILBSetLightProjectedTexture Sets a projected texture or a gobo for the spot light. X in light space maps to U in texture space, Z in light space maps to V in texture space.

Directional Light

The directional light is created using ILBCreateDirectionalLight It is possible to control its behavior using the functions:
ILBSetShadowAngle Sets how large angle of the sky the directional light covers.

Area Light

The area light is created using ILBCreateAreaLight
The scaling of the transform matrix is used to change the size of the light source. If the scaling is not changed it is a square extending from (-1, -1) to (1, 1).

Window Light

The window light is created using ILBCreateWindowLight
The scaling of the transform matrix is used to change the size of the light source. If the scaling is not changed it is a square extending from (-1, -1) to (1, 1). It is possible to control its behavior using the functions:
ILBSetShadowAngle sets the angle of the sky the window light covers, also affects the shadow from the virtual window that cuts out the window light.

3.8.3 Light Links

Light links is a way of making the instance only be affected by some of the lights in the scene. For a complete overview of light links, refer to the Light Links section.

3.8.4 Thread Safety

You should only operate on the same light source from one thread at the time.

3.8.5 Declaration

The Light functions are found in the file beastlightsource.h

3.9 Cameras

To create cameras, use the ILBCreate*Camera family of functions found in beastcamera.h. The transformation matrix parameter is used to place and orient the camera. A camera is directed in the negative Z direction by default. Supplying a matrix with translation and rotation can change that. There are two main types of cameras in Beast, Perspective Cameras and Environment cameras.

3.9.1 Perspective Camera

The perspective camera is created using the function ILBCreatePerspectiveCamera
You can control its FOV using the function ILBSetFov

3.9.2 Environment Camera

Environment Cameras renders environment maps. They are created using the function ILBCreateEnvironmentCamera. You can select different projections using the ILBEnvironmentCameraType enum parameter to the construction call.

3.9.3 Declaration

The Camera functions are found in the file beastcamera.h

3.10 Materials

The material model in the Beast API is technically Phong with the following channels:

- Diffuse
- Specular
- Emissive
- Specularity
- Transparency

Each channel can either be a constant color, a texture reference or vertex colors. To set a constant color for a channel use ILBSetMaterialColor. To set a texture for a channel use ILBSetMaterialTexture. To set usage of vertex colors use ILBSetMaterialUseVertexColors . Each channel is also associated with a scalar scale factor which is multiplied with the color. The scalar is set to 1 by default and can be controlled with the function ILBSetMaterialScale.

3.10.1 Transparency

The transparency channel is a color texture and it is defined in the opposite way compared to a typical alpha channel, i.e if you set the Transparency to RGB(1, 0, 0) it will make the material transparent for the red part of the light computation. It will also cast red colored shadows. For standard alpha behavior, use the function ILBSetAlphaAsTransparency. It will use whatever is set as alpha channel in the diffuse channel as a classic alpha channel, i.e 1 means opaque and 0 means black. It will not give any color to shadows but only give monochrome shadow casting.

3.10.2 Reflectivity

The strength of reflections are computed as Specular * Reflectivity. This means that mapping the specular channel will affect reflections as well. The reflectivity is a scalar value and is only controlled by ILBSetMaterialScale.

3.10.3 Shininess

Shininess sets the exponent in the specular computations. The Shininess is of Phong type. Note that shininess is a scalar value rather than a color and is completely controlled by ILBSetMaterialScale.

3.10.4 The Formula

What is being computed for each light is:

$$(N \cdot L)D(1 - T) + S(R \cdot V)^s \tag{3.1}$$

To that reflections, transmission and emissive colors are added in this way:

$$E + TC_T + SRC_R \tag{3.2}$$

Where:
N is the normal of the surface.
L is the direction towards the light source.
D is the diffuse color of the fragment with scale multiplied.
T is the transparency of the fragment with scale multiplied.
E is the emissive color of the fragment with scale multiplied.
R is the reflected light direction.
V is the direction towards the viewer.
S is the specular color of the fragment with scale multiplied.
s is the shininess of the fragment.
C_T is the color of the transmitted ray.
R is the reflectivity of the fragment with scale multiplied.
C_R is the color of the reflection ray.
What should be noted is that the specular highlights, emissive and reflections are not affected by transparency.

3.10.5 Non-physical Materials

When working with global illumination non-physical materials that amplifies light can be problematic since they can drench the scene in light. To avoid this sure to use emissive and not large diffuse or specular components to make object glow and emit light. Clamping the material colors can also be a good idea. There is an option to do this automatically from the configuration XML. Refer to **clampMaterials** in the Beast Manual for more information on it.

3.10.6 Declaration

The Material handling functions are found in the file beastmaterial.h.

3.11 Point Clouds

A point cloud is a set of points. Each point is defined by a position in world space and a normal. The point cloud is a part of the scene. It is created with ILBCreatePointCloud. Data is added to the cloud with ILBAddPointCloudData. The function takes parameters **pointData** and **normalData** as parameters which are pointers to ILBVec3 arrays. The parameter **pointCount** specifies how many points to add to the cloud. When the cloud is complete it must be ended with ILBEndPointCloud before it can be used.

3.11.1 Declaration

The point cloud related functions are found in the file beastpointcloud.h.

3.12 Render Passes

3.12.1 Relation between jobs, passes and targets

A job in the Beast API refers to an execution unit in which render passes and render targets is created. A render target describes what should be rendered (i.e. a camera view, a texture or a set of vertices). A render pass describes how it should be rendered (i.e. with full shading, with illumination only or with ambient occlusion). A pass is connected to a target by adding the pass to the target with ILBAddPassToTarget. One pass can be added to several different targets and one target can have several different passes added to it. They pass and targets always need to belong to the same job. When the job is executed all targets will get its respective connected passes rendered.

3.12.2 Normalization

By default Beast will render images with full dynamic range. If low dynamic range images (8 bits per channel) are desired Beast can scale each entity to fit the 0-255 range automatically when baking and the original range can be retrieved. This normalization is enabled for each render pass with ILBNormalizeTextures. If the **perChannel** parameter is true then each channel (i.e. red, green and blue) will be normalized separately. If set to false all channels of the pass will be normalized together. When the pass is done the normalization values for each entity can be retrieved with ILBGetNormalizationData. The parameter **pass** specifies which pass to retrieve normalization data for. If normalization per channel is used the parameter **channel** will specify which channel to retrieve normalization data for. The function is called once for each channel when operating in this mode. If using global normalization channel can be set to 0. The minimum and maximum value of the framebuffer before normalization is stored in the pointers **minValue** and **maxValue**.

3.12.3 Types

Full Shading Pass

The full shading pass is created with ILBCreateFullShadingPass. The output will be 5 channels in the form of:

$$(RGBA)Z$$

Where Z is the Z channel

Illumination Pass

The illumination pass is created with ILBCreateIlluminationPass. The parameter **mode** sets the ILBIlluminationMode of the pass which can have the following values:

- ILB_IM_DIRECT_ONLY Only direct light.
- ILB_IM_INDIRECT_ONLY Only indirect light.
- ILB_IM_FULL All incoming light.

The output from the illumination pass is 5 channels in the form of:

$$(RGBA)Z$$

Where Z is the Z channel

RNM Pass

The RNM pass is created with ILBCreateRNMPass. The following parameters are of interest:

- **mode** sets the ILBIlluminationMode which works like in the illumination pass.
- **samples** is the number of samples for non-adaptive RNM. When it is set to 0 samples the pass will be adaptive which is recommended.
- **basis** selects the RNM basis vectors from ILBRNMBasis. The supported basis vector configurations are:
  ILB_RB_HL2 Half-Life 2 compatible basis
  ILB_RB_UE3 Unreal Engine 3 compatible basis
  ILB_RB_UE3_FLIPPED Unreal Engine 3 basis in untouched order

Additionally the following functions is used to control the RNM pass:
ILBSetLambertianClamp enables a clamping value on the sum of the lambertian shading value of the RNM components. If enabled Beast will scale the result of the pass so that the sum of the lambertian shading equals the clamping value.
ILBSetAllowNegative sets how the RNM pass treats light which is under the horizon of the surface. In the ordinary illumination pass such a light would yield a light contribution of 0 but in the RNM pass it can still affect one or two of the RNM basis components. It also controls if negative RNM components will be clamped to 0 or if they are allowed to be negative. The supported states are

- ILB_AN_ALLOW Allows negative RNM values.
- ILB_AN_DISALLOW Clamps RNM values to 0.
- ILB_AN_DISALLOW_CULL_HORIZON Clamps RNM values to 0 and culls lights which i below the horizon of the surface (this is the default).

ILBIncludeNormalComponent makes the RNM pass include illumination on the surface normal as well as the three RNM basis vectors.
ILBRNMMatchNormalIntensity scales the RNM components to match the intensity of the normal component.
The output of the RNM pass will be in the form of (RGBA) for each RNM basis and additionally once for the normal component. If the normal component is enabled the output will have 17 channels in this configuration:

$$(RGBA)_1(RGBA)_2(RGBA)_3(RGBA)_NZ$$

Where:
1 is the first RNM basis vector.

2 is the second RNM basis vector.
3 is the third RNM basis vector.
N is the normal component.
Z is the Z channel.

If the normal component is not included the pass will have 13 channels.

Light Pass

The light pass is created with ILBCreateLightPass. It takes as parameter **type** of the type ILBLightPassType which specifies the output of the light pass. It can have the following values:

- ILB_LP_LIGHTMAP - Stores the incoming light.
- ILB_LP_SHADOWMAP - Stores a value proportional to the light source intensity.
- ILB_LP_FULLSHADING - Stores the full shading value.

When using ILB_LP_SHADOWMAP has an additional render mode called signed distance field. It is enabled with ILBEnableSignedDistanceField with the parameters **pixelFilterSize** and **maxWorldDistance** controlling the size of the filter and the max distance. More information on signed distance fields can be found in the Beast Manual.
The light pass is used to bake with only a specific set of light sources affecting each entity in the map. Each specific set is called a light pass entry and is created with ILBCreateLightPassEntry. The light pass could be used to create static shadow map textures. The light pass is controlled by defining one or more light sets which each affects one or more target entities. This means that one can for example bake an instance twice in the same texture, lit by light A in the first entity and lit by light B in the second. Light sources are added to the light pass entry with ILBAddLightToPass and target entities are added with ILBAddTargetToPass.

This is how the example would be set up:

```
ILBRenderPassHandle lightPass;
ILBLightPassEntry lpEntry1, lpEntry2;
ILBCreateLightPass(job, "LightPass", ILB_LP_SHADOWMAP, &lightPass);
ILBCreateLightPassEntry(lightPass, &lpEntry1);
ILBAddTargetToPass(lpEntry1, targetEntity1);
ILBAddLightToPass(lpEntry1, lightA);
ILBCreateLightPassEntry(lightPass, &lpEntry2);
ILBAddTargetToPass(lpEntry2, targetEntity2);
ILBAddLightToPass(lpEntry2, lightB);
```

The output from the light pass is 5 channels in the form of:

$$(RGBA)Z$$

Where Z is the Z channel

Ambient Occlusion Pass

The ambient occlusion pass is created with ILBCreateAmbientOcclusionPass. The following parameters are of interest:

- **maxDistance** The maximum distance to count as occlusion. A setting of 0 equals infinity.
- **coneAngle** The angle of the cone to consider for occlusion, in degrees. The default is 180.

Additionally the following functions control the behaviour of the ambient occlusion pass:

- ILBSetAOAdaptive Makes the occlusion pass use adaptive sampling. This will increase the speed considerably.
- ILBSetAONumRays Sets the minimum and maximum number of occlusion rays to sample for each point.
- ILBSetAOContrast Changes the contrast and scale of the occlusion pass.
- ILBSetAOUniformSampling Changes the weighting of the occlusion samples to be uniform instead of instead of cos-weighted which is the default.
- ILBSetAOSelfOcclusion Sets how the pass reacts to self occlusion. The input is a member of the enum ILBAOSelfOcclusion. The default is ILB_SO_ENABLED which means that the pass considers self occlusion as any other occlusion.
- ILBEnableBentNormals Enables output of bent normals. The bent normal is the direction of least occlusion in the hemisphere centered around the normal of each point sampled. When enabled the R, G and B component of the output will represent the X, Y and Z of the bent normal in world space. The bent normal is normalized and mapped from the range [-1..1] to a [0..1] range.

The output from the occlusion pass is 5 channels in the form of:

$$(RGBA)Z$$

Where Z is the Z channel

LUA Pass

The Lua pass is created with ILBCreateLuaPass. It takes as input **scriptFile** which is the path of a Lua script controlling the baking of the pass. The output from the Lua pass will contain as many channels as specified in the Lua bake script. The channel count of a frame buffer or vertex buffer can be found with ILBGetChannelCount. For more information on Lua baking, please refer to the Beast Manual.

3.13 Render Targets

3.13.1 Target Types

Camera Target

The purpose of the camera target is to render a camera view of the scene. It is created with ILBCreateCameraTarget. The parameter **camera** sets which camera to render from. The parameters **width** and **height** sets the dimensions of the framebuffer of the target. The only supported render pass on the camera target is the full shading pass.

Texture Target

The purpose of the texture target is to render (bake) one or more instances mapped on a framebuffer using UV coordinates. It is created with ILBCreateTextureTarget. The parameters **width** and **height** sets the dimensions of the framebuffer of the target. Instances to bake are added to the texture target with ILBAddBakeInstance, which will create a target entity $ILBTargetEntityHandle$. The UV set to use for baking the entity can be changed with ILBSetBakeUVSet. The entity can also have an offset and a scale applied to the UV coordinates through the function ILBSetUVTransform.
All render passes are supported on the texture target.

Atlased Texture Target

The atlased texture target is in many ways similar to the ordinary texture target but instead of placing each instance explicitly in the UV map it will automatically pack the instances together in one or more frame buffers. The atlased texture target is created with ILBCreateAtlasedTextureTarget. The parameters **maxWidth** and **maxHeight** specifies the *maximum* dimensions of the framebuffer. If for example the maximum framebuffer dimensions is set to 1024x1024 but only 512x256 is needed then the framebuffer will be 512x256. The parameter **maxTextures** controls the maximum number of framebuffers to be generated. If set to 0 there is no limit. There are two functions controlling the placement of the entities in the UV map, ILBSetAtlasAlignment and ILBSetAtlasPadding. Setting the alignment to a value larger than one will make sure that each entity only is placed on every n:th texel in the UV map where n is the alignment. Setting the padding will make sure that there are at least m empty texels between the entities, where m is the padding. Instances are added to the atlased texture target in the same fashion as the texture target, with ILBAddBakeInstance. The desired resolution for an entity can be forced with ILBSetBakeResolution. Another way to influence the resolution of the entity is ILBSetTexelScale, where the relationship between world space and texture coordinates is specified. When the target is finished the function ILBGetAtlasInformation can be used to find out in which frame buffer and where the entity was placed by the atlased layout. There is also a spatial packing feature which tries to group entities in lightmaps based on their respective proximity in world space. It can be enabled with ILBEnableAtlasSpatial
All render passes are supported on the atlased texture target.

Vertex Target

The purpose of the vertex target is to evaluate a render pass for each vertex in an instance. It is created with ILBCreateVertexTarget. Instances are added in the same way as the other baking targets, with ILBAddBakeInstance.
All render passes are supported on the vertex target.

Point Cloud Target

The purpose of the point cloud target is to evaluate a render pass for each point in a set of point (these sets of points are called point clouds). The target is created with ILBCreatePointCloudTarget. Point clouds are added to the target with ILBAddBakePointCloud.
The only supported render pass on the point cloud target is the Lua Pass.

3.14 Reading back the output

The output of a target/render pass combination is stored in a frame buffer or vertex buffer, or in the case of the atlased texture target one or more frame buffers. The camera, texture and atlased texture targets output frame buffers and the vertex and point cloud target output vertex buffers.

3.14.1 Frame Buffers

To retrieve a frame buffer ILBGetFramebuffer is used, with the parameters **target** and **pass** specifying the target and the render pass. For an atlased texture target the parameter **index** is used to specify which framebuffer to get. The frame buffer contains a number of channels, depending on the render pass used. ILBGetChannelCount is used to get how many channels a certain frame buffer contains. ILBGetChannelName is used to get the name of each channel. The resolution of the frame buffer can be found with ILBGetResolution. The contents of the frame buffer is read with ILBReadRegionHDR or ILBReadRegionLDR. The data is originally stored with high dynamic range and the LDR function can be used to retrieve it in a low dynamic range space with a given gamma. When the work on the frame buffer is completed it can be destroyed with ILBDestroyFramebuffer. Otherwise it is destroyed when the job it belongs to is destroyed.

3.14.2 Vertex Buffers

The vertex buffer works like a one dimensional frame buffer where the Y height always is 1. The following functions works in the same way as for frame buffers:

- ILBGetChannelCount
- ILBGetChannelName
- ILBGetResolution - **height** will always be 1 though.

- ILBReadRegionHDR - **minY** needs to be 0 and **maxY** needs to be 1.

Note that ILBReadRegionLDR is not supported on vertex buffers.

3.15 Jobs

The job object is responsible for controlling and monitoring the invocation of Beast. It is created from a Beast Manager object with ILBCreateJob and is defined by a Scene object and an XML file with the render settings.

3.15.1 Starting the Job

The directory where the output files from the job will end up can be specified with ILBSetJobOutputPath. If this is not specified a directory will be created in the cacheDirectory hierarchy. The job can then be started with ILBStartJob. There are three different modes for how the result of the job is displayed, described by the enum ILBShowResult:

- ILB_SR_NO_DISPLAY: No render window is shown, the process will exit when done.
- ILB_SR_CLOSE_WHEN_DONE: A render window will show the progress of the rendering. The window will close and the process will exit when the job is done.
- ILB_SR_KEEP_OPEN: A render window will show the rendering and stay open until the user closes it. This mode is intended for showing camera renders to the user.

You can also specify whether the job should be rendered distributed or locally.

3.15.2 Distribution

The Beast API has experimental support for distribution using XGE. The Beast API will use distribution for the passes that support it, which is currently:

- Lua Pass
- RNM Pass
- Illumination Pass
- Full Shading Pass

To use it, make sure XGE is present in your path environment. The last parameter to ILBStartJob sets how distribution should be used. The valid values are

- ILB_RD_FORCE_LOCAL: The rendering will never use distribution.
- ILB_RD_AUTODETECT: This option tries to detect whether it is possible to render distributed and fall back to local rendering otherwise.
- ILB_RD_FORCE_DISTRIBUTED: This option means that the rendering will only render distributed. If it can not find the necessary executables the rendering will fail.

3.15.3 Monitoring Progress

While the Beast Job is running the client needs to call ILBWaitJobDone at regular intervals. The function will return when either:

- The job is done
- There is new progress
- The timeout has expired

If the timeout is set to 0 the call will return immediately. If timeout is 0xffffffff it will wait until the job is done or there is new progress information available. When ILBWaitJobDone has returned the client can check if it is completed with ILBIsJobCompleted. When the job is completed the result of the job is available with ILBGetJobResult. The job can also be aborted with ILBCancelJob.
To get information about the progress of the job ILBJobHasNewProgress is used. If the function indicates that either a new activity has started or there is new progress information on the current activity ILBGetJobProgress can be used to get a string describing the current activity and a completion percentage.

3.15.4 ILBExecuteBeast

If you do not need progress updates or asynchronous job running there is a function ILBExecuteBeast which encapsulates everything about running a job. It will take a job as input, start it, wait for it to finish and then return. It can be used like this:

```
ILBJobHandle job;
ILBCreateJob(bm, "job1", scene, "settings.xml", &job);
// Create target(s) and pass(es) on the job.
ILBJobStatus jobstatus;
ILBExecuteBeast(bm, job, ILB_SR_KEEP_OPEN, ILB_RD_FORCE_LOCAL, &jobstatus);
```

3.15.5 Examples

This is a minimal example of creating and starting a job and waiting for it to finish. Please note that error handling is omitted for clarity.

```
ILBJobHandle job;
ILBJobStatus jobstatus;
ILBBool isRunning = 1;

ILBCreateJob(bm, "job1", scene, "settings.xml", &job);

ILBTargetHandle cameraTarget;
ILBCreateCameraTarget(job, "cameraTarget", camera, 640, 480, &cameraTarget);

ILBRenderPassHandle fullShadingPass;
ILBCreateFullShadingPass(job, "fullShading", &fullShadingPass);
```

```
ILBAddPassToTarget(cameraTarget, fullShadingPass);

ILBStartJob(job, ILB_SR_CLOSE_WHEN_DONE, ILB_RD_FORCE_LOCAL);

while (isRunning) {
  ILBWaitJobDone(job, 0xffffffff);
  ILBIsJobRunning(job, &isRunning);
}
ILBGetJobResult(job, &jobstatus);
ILBDestroyJob(job);
```

3.15.6 Thread Safety

The client may start several jobs and run them in parallel from different threads without worrying about locking.

3.15.7 Declaration

The Job handling functions are found in the file beastjob.h

3.16 Light Links

Light links is used to control what light sources affects what objects.

3.16.1 Light Centric vs Object Centric

There are two types of light links, light centric and object centric, meaning that either you specify what instances are being affected by a light source or what light sources affect an instance.
Object centric light links are specified using the function ILBAddInstanceLightLinks, light centric are specified using ILBAddLightLightLinks.

3.16.2 Inclusive vs Exclusive

It is also possible to specify light links that are inclusive or exclusive. Setting an inclusive light link from a light to an instance means that this light will affect only this object, an exclusive means that it affect all objects but the one it was linked to. One instance or light source can only have exclusive or inclusive light links, but not both at the same time.

3.16.3 Conflicts

When the light links of a light source and instance says different things, the light source light link will be the one used. To avoid bothering about this consider using only one way of specifying light links. Lights and instances without light links will affect each other by default.

3.17 Using Beast from other languages

When using the Beast API from a language that cannot use the header files and the import library you need to manually load the dynamic library and import the the API functions. Exactly how this is done depends on the programming language. This information also applies if loading the dynamic library manually from C/C++ using LoadLibrary rather than using the import library.

There is some magic happening automatically in C/C++ when it comes to choosing the string encoding and verifying that the header has the right version. That needs to be taken care of manually when using other languages. The file beastmaster.h contains two static functions, ILBSetLogTarget and ILBCreateManager. They automatically calls the function ILBSetStringEncodingImp before calling ILBSetLogTargetImp and ILBCreateManagerImp respectively.

It makes sure the string encoding to be used is setup before calling any other function. It is supported to call ILBSetStringEncodingImp multiple times as long as you always give the same encoding. To get this working properly ensure that ILBSetStringEncodingImp is the first function called in the wrapper as well.

The other thing happening is the value ILB_BEAST_INTERFACE_VERSION is added to the parameter list for ILBCreateManager when calling ILBCreateManagerImp. This gives the DLL the ability to check whether the dynamic library has the same version as the header file or language wrapper trying to load it. ILB_BEAST_INTERFACE_VERSION is a counter that is being updated every time a version of the API with changes in the interface is being released.

The rest of the API calls should be straight forward to import into any language. We hope to deliver wrappers for some common languages in the future. If you are interested in contributing one, let us know!

Appendix A

Vectors and Coordinate Systems

Every 3d engine has its own way of setting up coordinate systems and handling vectors. This part is not meant as a math reference, but a help for developers to make sure they pass in data in a way that makes sense in Beast. Refer to a Linear Algebra book or a book on rendering such as [1] for more on this subject.

A.1 Vectors and Matrices

In the Beast API Vectors are treated as columns. It has a transformation pipeline based on homogeneous coordinates (4x4 Matrices) it only uses 3 components for vectors and points, the W component is implicitly specified depending on if it's a vector or a point. Exactly what it is should be obvious from the context it's being used in.

Vector matrix multiplications is expressed in the following manner:

$$\begin{pmatrix} v'_x \\ v'_y \\ v'_z \\ v'_w \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{21} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix} \times \begin{pmatrix} v_x \\ v_y \\ v_z \\ v_w \end{pmatrix}$$

This gives the following layout for the most common matrices:

Translation

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Counter clockwise rotation around X axis

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Counter clockwise rotation around Y axis

$$\begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Counter clockwise rotation around Z axis

$$\begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A.2 Coordinate Systems

In the Beast API all objects with a position and an orientation in the world are specified with a matrix. This may seem redundant for primitives such as directional lights or point lights, but in the end it means that the same transformation pipeline applies to everything.
Beast uses a Right Handed Coordinate system like OpenGL. DirectX uses a left handed coordinate system so data authored for DirectX will need to flip the index order when specifying triangles to make sure the normals point in the right direction.

A.3 Camera Coordinates

The Camera coordinate system is similar to the one in OpenGL, an identity camera transform the X axis to the right, the Y axis up and negative Z forward.

A.4 Light Coordinates

Directed Light sources are defined to point in the negative Y direction when defined with an identity matrix.

A.5 Screen Coordinates

Beast don't use projection matrices to go from world coordinates to screen coordinates, but camera objects. These are composed of a matrix to orient them in the world. The exact definition of screen coordinates differs from each camera.

A.6 Texture Coordinates and Images

Beast images and textures are treated as a coordinate system where origin is the lower left corner and up is positive Y and right is positive X. Note that many image formats defines the origin in the upper left corner with positive Y pointing down.

Appendix B

Gamma Correction

Colors in Beast are expressed in a linear space. This is different from many programs that work with gamma corrected colors.
Historically gamma comes from a physical phenomena in CRT monitors giving a non linear intensity Typically the color outputted is $C_{out} = (C_{image})^{\gamma}$. γ is generally somewhere between 1.5 and 2.5. It might seem like a clumsy convention but it happens to be close to the inverse of how the human eye perceives light. This means that it's a good way of encoding colors to use the bits in the pixels as efficiently as possible. Encoding colors in 8 bits per pixel with gamma 1 (i.e linear colors) causes banding in the darker part of the images and wastes bits in the brighter parts. The way to go to linear from a gamma encoded color space is done like this:

$$\begin{cases} R_{linear} = (R_{gamma})^{\gamma} \\ G_{linear} = (G_{gamma})^{\gamma} \\ B_{linear} = (B_{gamma})^{\gamma} \end{cases} \tag{B.1}$$

Going back to gamma corrected colors is done this way:

$$\begin{cases} R_{gamma} = (R_{linear})^{1/\gamma} \\ G_{gamma} = (G_{linear})^{1/\gamma} \\ B_{gamma} = (B_{linear})^{1/\gamma} \end{cases} \tag{B.2}$$

Note that Alpha channels are not gamma corrected but treated as linear.
Exactly what γ is depends on the screen and the color data but a good start value is 2.2 which is somewhat of a standard for PC monitors.

B.1 How does gamma affect a developer?

Except for the fact that there are different gamma on different screens that can cause trouble for artists, mathematical operations doesn't give the correct results when working with gamma corrected colors. Consider for example:

$$C^{\gamma} + D^{\gamma} \neq (C + D)^{\gamma} \tag{B.3}$$

Doing math in gamma space can give rise to many problems such as incorrect colors and brightness. It's getting even worse when doing global illumination with multiple light bounces where the errors amplify.
The right way is to make sure all colors authored for the screen (typically what you find in an 8 bit texture or a color picked using a color picker) is converted to linear space at some point before doing computations on them and converting them back to gamma space before displaying them.
Exactly how to get this right depends on the engine, the editor and the lighting pipeline used. It needs to care about proper conversions and making sure images are encoded in formats using their bits efficiently.
Beast treats input colors as linear colors so if working with gamma corrected colors they need to be converted before stored in as ILBLinearRGB or ILBLinearRGBA.
Another problem to consider is that the gamma computations are really just meant for color components in the span 0-1. Naively converting a gamma corrected color component with the value 20 to linear will give an intensity of $20^{2.2}$ which is about 728, perhaps not what the artist expected.
A good start to handle this is separating intensity and Color in settings. Good sources for information about gamma correction are [1], [2] and [3].

Appendix C

XML Settings

When using the Beast API, there is still some settings that need to be controlled through the use of an XML settings file. These settings are global for each job. Beast is very tweakable, which means that there are a lot of settings that are exposed to the user. Fortunately, there is no need to alter all settings every time. This document focuses on the most important settings, what they do and how they relate to other settings. For a more reference style rundown, please use the Beast manual. If you feel that something is missing, please let us know at support@illuminatelabs.com.
There are three main areas this manual intends to explain:

1. Sampling Settings: This part explains the sampling schemes used by Beast. It also contains information about the post filters which can be used when texture baking.
2. Environment Settings: How to setup Beast to use HDR images or colors as Environment maps.
3. Global Illumination: Explains how the Global Illumination should be setup in Beast.

Also note that some settings have more functions/values than presented here. This document focuses on the most common scenarios.

C.1 Sampling Settings

C.1.1 Adaptive Sampling (Texture)

Beast uses an adaptive sampling scheme when sampling light maps. The light must differ more than a user set contrast threshold for Beast to place additional samples in an area. The sample area is defined by a Min and Max sample rate. The user sets the rate in the -4..4 range which means that Beast samples from 1/256 sample per pixel to 256 samples per pixel (the formula is 4ŝamplerate).
It is recommended to use at least one sample per pixel for production use (Min sample rate = 0). Undersampling is most useful when doing camera renders or baking textures with big UV-patches. When Beast has taken all necessary samples for an area, the final pixel value is

weighed together using a filter. The look the filter produces is dependent on the filter type used and the size of the filter kernel. The available filters are:

- Box. each sample is treated as equally important. The fastest filter to execute but it gives blurry results.
- Triangle. The filter kernel is a tent which means that distant samples are considered less important.
- Gauss Filter. Uses the Gauss function as filter kernel. This gives the best results (removes noise, preserves details).

There are more filters available, but these three are the most useful. The kernel (filter) size is given in pixels in the range 1..3. Beast actually uses all sub pixels when filtering, which yields better results than doing it afterwards in Photoshop.

XML

```
<AASettings>
  <!--Turns on Adaptive Super sampling-->
  <samplingMode>Adaptive</samplingMode>
  <!--Sets the min sample rate, for baking 0 (ie one sample per pixel) is
      recommended-->
  <minSampleRate>0</minSampleRate>
  <!--Sets the max sample rate, the formula used is 4^maxSampleRate (1,
      4, 16, 64, 256 samples per pixel)-->
  <maxSampleRate>0</maxSampleRate>
  <!--The contrast value which controls if more samples are necessary, a
      lower value forces more samples-->
  <contrast>0.1</contrast>
  <!--Sets which filter type to use. Most useful ones for Baking are Box,
      Triangle and Gauss-->
  <filter>Gauss</filter>
  <!--Sets the filter size in pixels. Set in the 1..3 range-->
  <filterSize><x>1.0</x><y>1.0</y>
</AASettings>
```

C.1.2 Adaptive Sampling (Vertex)

When baking to vertices, super sampling is achieved by subdividing each triangle and then accumulate all samples to the original triangle. The sampling level is based on how big the triangle is in world space and which subdivision step size that is used. If the subdivision step is left at 0, Beast will automatically calculate sample levels between the min and max sample levels. Otherwise, the subdivision step size can be set to the "resolution" to sample the vertex data at. The actual samples taken are randomly distributed over the triangle and then accumulated to each vertex. The default min sample value is 2, which means that small triangles are under sampled.

XML

```
<VertexBakeSettings>
  <VertexBakeSamplingOptions>
    <!--Use triangle super sampling. For one sample per vertex, set
        PerVertex-->
    <Type>TriangleSubdiv</Type>
    <!--The minimum number of samples to take per triangle-->
    <MinSamples>2</MinSamples>
    <!--The maximum number of samples to take per triangle-->
    <MaxSamples>2</MaxSamples>
    <!--The resolution in which to sample in world space, a low value
        gives more samples-->
    <SubDivStepSize>0</SubDivStepSize>
  </VertexBakeSamplingOptions>
</VertexBakeSettings>
```

C.2 Texture Specific Settings

- Edge dilation is used to "expand" the baked light map. If this is not done there may be black borders when applying the maps in your engine since the GPU will filter in empty texels.
- Bilinear Filter is used to make sure that the data in the lightmap is "correct" when the GPU applies bilinear filtering. This is most noticable when the atlases are tightly packed. If there is only one pixel between two different UV patches, the bilinear functionality in Beast will make sure the that pixel is filled with the color from the correct patch. This minimizes light seams.
- Conservative Rasterization is used when the UV-chart does not cover the entire pixel. If such a layout is used, Beast may miss the texel by mistake. If conservative rasterization is used Beast will guarantee that it will find a UV-layout if present. Note that Beast will pick any UV-layout in the pixel. Conservative Rasterization often needs to be turned on if the UV atlases are tightly packed in low resolutions or if there are very thin objects present.

XML

```
<TextureBakeSettings>
  <!--Expands the rendered region with the number of pixels specified-->
  <edgeDilation>5</edgeDilation>
  <!--Turns on bilinear filtering, always use if charts are tightly
      packed-->
  <bilinearFilter>true</bilinearFilter>
  <!--Use if you require sub pixel resolution to find UV Layouts-->
  <conservativeRasterization>false</conservativeRasterization>
</TextureBakeSettings>
```

C.3 Environment Settings

The environment settings in Beast control what happens if a ray misses all geometry in the scene. The environment can either be a constant color or an HDR image in lat-long format. Note that environments should only be used for effects that can be considered to be infinitely far away, meaning that only the directional component matters. The environment settings have separate entries for Global Illumination and Camera (ray traced) rays. Beast works this way to accommodate the need for higher resolution source images to get good looking background and reflective effects compared to the rather low frequency needed for Global Illumination effects. To bake light maps typically only the giEnvironment setting is used, since this is the environment that the GI system can see. For camera renders, the rtEnvironment can be used to make the environment visible in the background and in reflections. Usually either SkyLight or IBL is enabled for Global Illumination renders. The SkyLight is a simple constant color, but is a easy way to add diffuse indirect lighting, and can give very pleasing results. The color is specified by skyLightColor. It is often a good idea to keep the color below 1.0 in intensity to avoid boosting by gamma correction. Boost the intensity instead with the giIntensity setting. Setting the environment to IBL requires the path to an HDRI image to be entered in the iblImageFile setting (it can be in either HDR or EXR format). Again, use giIntensity to boost the intensity.

XML

```
<EnvironmentSettings>
  <!--The type of Environment. None, Skylight (constant color) or IBL
      (texture)-->
  <giEnvironment>SkyLight</giEnvironment>
  <rtEnvironment>SkyLight</rtEnvironment>
  <!--A scale factor for the intensity, used for avoiding gamma
      correction errors and to scale HDR textures to something that fits
      your scene-->
  <giIntensity>1.0</giIntensity>
  <rtIntensity>1.0</rtIntensity>
  <!--A constant environment color. Used if type is Skylight-->
  <skyLightColor>
    <r>0.7</r>
    <g>0.7</g>
    <b>0.7</b>
    <a>1.0</a>
  </skyLightColor>
  <!--The image file to be used in hdr or OpenEXR format. The file should
      be long-lat. Used if type if IBL-->
  <iblImageFile>filename.exr</iblImageFile>
</EnvironmentSettings>
```

C.4 Global Illumination

C.4.1 Final Gather

Quality Settings

The settings below control the quality or correctness of the Final Gather solution. They are modified to suit preview and production settings. The normal usage scenario is this:

1. For each baking set up Contrast Threshold and Number of Rays may be adjusted.

There are no perfect settings for these since they depend on the complexity of the geometry and light setup.

2. Check Visibility and Light Leakage reduction are expensive operations and should only be used to remedy actual light leakage problems. These settings will only help if the light leakage is caused by the Global Illumination calculations. A very common light leakage situation is occurs with a wall as a single plane with no thickness. The light leaking through in that situation does not come from GI.

3. Gradient threshold should only be changed if there are white halos around corners.

Step 2 and 3 should not need much tweaking in most scenes.

- Contrast Threshold: Controls how sensitive the final gather should be for contrast differences between the points during precalculation. If the contrast difference is above this threshold for neighbouring points, more points will be created in that area. This tells the algorithm to place points where they are really needed, e.g. at shadow boundaries or in areas where the indirect light changes quickly. Hence this threshold controls the number of points created in the scene adaptively. Note that if a low number of final gather rays are used, the points will have high variance and hence a high contrast difference. In that the case contrast threshold needs to be raised to prevent points from clumping together or using more rays per sample.

- Number of Rays: The maximum number of rays taken in each Final Gather sample. More rays gives better results but take longer to evaluate.

- Use Cache: Sets which type of information the cache stores. For the RNM pass it should be "RadianceSH", otherwise when baking standard light maps it should be "Irradiance". For RNMs the Beast API will set this up for you. However, if the "gather.useRadianceCache" LUA bake script feature is used it needs to be enabled manually in the XML.

- Check Visibility: Turn this on to reduce light leakage through walls. When points are collected to interpolate between, some of them can be located on the other side of geometry. As a result light will bleed through the geometry. To prevent this Beast can reject points that are not visible.

- Check Visibility Depth: Controls for how many bounces the visibility checks should be performed. Adjust this only if experiencing light leakage when using multi bounce Final Gather.

- Light Leak Reduction: This setting can be used to reduce light leakage through walls when using final gather as primary GI and path tracing as secondary GI. Leakage, which can happen when e.g. the path tracer filters in values on the other side of a wall, is reduced by using final gather as a secondary GI fallback when sampling close to walls or corners. When this is enabled a final gather depth of 3 will be used automatically, but the higher depths will only be used close to walls or corners. Note that this is only usable when path tracing is used as secondary GI.

- Light Leak Radius: Controls how far away from walls the final gather will be called again, instead of the secondary GI. If 0.0 is used Beast will try to estimate a good value. If this does not eliminate the leakage it can be set to a higher value manually.

- Normal Threshold: Controls how sensitive the final gather should be for differences in the points normals. A lower value will give more points in areas of high curvature.
- Gradient Threshold: Controls how the irradiance gradient is used in the interpolation. Each point stores its irradiance gradient which can be used to improve the interpolation. In some situations using the gradient can result in white "halos" and other artifacts. This threshold can be used to reduce those artifacts (set it low or to 0).
- Accuracy (vertex bake): The vertex baking is using another version of the Final Gather filter. Here the Accuracy can be used to control the number of samples created. Usually it should be set a bit lower than the default 1.0 when vertex baking.

Look Settings

These settings affect the look of the lighting and should be set on a per project / per level basis.

Final Gather Depth

Controls the number of indirect light bounces. A higher value gives a more correct result, but the cost is increased rendering time. For cheaper multi bounce GI, use Path Tracer as the secondary integrator instead of increasing depth.

Final Gather Attenuation

This can be used to add a falloff effect to the final gather lighting. When fgAttenuationStop is set higher than 0.0 this is enabled.

- Attenuation Start: The distance where attenuation is started. There is no attenuation before this distance.
- Attenuation Stop: Sets the distance where attenuation is stopped (fades to zero). There is zero intensity beyond this distance. To enable attenuation set this value higher than 0.0. The default value is 0.0.
- Falloff Exponent: This can be used to adjust the rate by which lighting falls off by distance. A higher exponent gives a faster falloff.

Ambient Occlusion

If Final Gather is used with multiple depths or with Path Tracing as Secondary GI the result can become a bit "flat". A great way to get more contrast into the lighting is to factor in a bit of ambient occlusion into the calculation.

- AO Influence: Blend the Final Gather with Ambient Occlusion. Range between 0..1. 0 means no occlusion, 1 is full occlusion.

- AO Max Distance: Max distance for the occlusion rays. Beyond this distance a ray is considered to be unoccluded. Can be used to avoid full occlusion for closed scenes such as rooms or to limit the AO contribution to creases.
- AO Contrast: Can be used to adjust the contrast for ambient occlusion.
- AO Scale: A scaling of the occlusion values. Can be used to increase or decrease the shadowing effect.

C.4.2 Path Tracing

Use path tracing to get fast multi bounce global illumination. It should not be used as primary integrator for baking since the results are quite noisy which does not look good in light maps. It can be used as primary integrator to adjust the settings, to make sure the cache spacing and accuracy is good. The intended usage is to have it set as secondary integrator and have single bounce final gather as primary integrator. Accuracy and Point Sizecan be adjusted to make sure that the cache is sufficiently fine grained. Camera renders is a useful tool for tweaking the parameters.

- Accuracy: Sets the number of paths that are traced for each sample element (pixel, texel or vertex). For preview renderings, a low value like 0.5 to 0.1 can be used. This means that 1/2 to 1/10 of the pixels will generate a path. For production renderings values above 1.0 may be used, if necessary to get good quality.
- Point Size: Sets the maximum distance between the points in the path tracer cache. If set to 0 a value will be calculated automatically based on the size of the scene. The automatic value will be printed out during rendering, which is a good starting value if the point size needs to be adjusted.
- Cache Direct Light: When this is enabled the path tracer will also cache direct lighting from light sources. This increases performance since fewer direct light calculations are needed. It gives an approximate result, and hence can affect the quality of the lighting. For instance indirect light bounces from specular highlights might be lost.
- Check Visibility: Turn this on to reduce light leakage through walls. When points are collected to interpolate between, some of them can be located on the other side of geometry. As a result light will bleed through the geometry. To prevent this Beast can reject points that are not visible. Note: If using this turn off light leakage reduction for Final Gather.

XML

```
<GISettings>
  <enableGI>true</enableGI>

  <!--Setup which integrators to use and their intensity and saturation-->
  <primaryIntegrator>FinalGather</primaryIntegrator>
  <secondaryIntegrator>PathTracer</secondaryIntegrator>
  <primaryIntensity>1.0</primaryIntensity>
  <primarySaturation>1.0</primaryIntensity>
```

```
<secondaryIntensity>1.0</secondaryIntensity>
<secondarySaturation>1.0</secondarySaturation>

<!--Final Gather Settings, Quality-->
<!--Lower value produces more samples. Make sure to use enough rays,
    otherwise points will cluster-->
<fgContrastThreshold>0.1</fgContrastThreshold>
<!--More rays gives better result-->
<fgRays>300</fgRays>
<!--Prevents light leakage when interpolating fg samples-->
<fgCheckVisibility>false</fgCheckVisibility>
<!--Adjust this if you want to check visibility on higher levels as
    well-->
<fgCheckVisibilityDepth>1</fgCheckVisibilityDepth>
<!--Helps to reduce light leakage caused by secondary integrator-->
<fgLightLeakReduction>false</LightLeakReduction>
<!--A radius in world space around the sample point where fg is used
    instead of Secondary integrator-->
<fgLightLeakRadius>0.0</fgLightLeakRadius>
<!--Lower this if you have white halos in your bake-->
<fgGradientThreshold>0.5</fgGradientThreshold>
<!--How much a normal can differ in the cache. Given as cos(a)-->
<fgNormalThreshold>0.2</fgNormalThreshold>
<!--For rnms, we want to store incoming light in a directions as an
    SH-->
<fgUseCache>RadianceSH</fgUseCache>

<!--Final Gather Settings, Look-->
<!--Increase this to get multi bounce GI secondary integrator is
    disabled-->
<fgDepth>1</fgDepth>
<!--World space distance-->
<fgAttenuationStart>0.0</fgAttenuationStart>
<!--World space distance, must be set to >0 for attenuation to be
    used-->
<fgAttenuationStop>0.0</fgAttenuationStop>
<!--Exponent controls the look of the falloff-->
<fgFalloffExponent>0.0</fgFalloffExponent>
<!--Set to >0 for AO to be calculated-->
<fgAOInfluence>0.0</fgAOInfluence>
<!--Set to <"room␣size" to avoid total occlusion-->
<fgAOMaxDistance>0.0</fgAOMaxDistance>
<!--Controls the look of the transition form white to black-->
<fgAOContrast>1.0f</fgAOContrast>
<!--Increase this to get more "punch" in the shadows-->
<fgAOScale>1.0</fgAOScale>

<!--Path tracer settings-->
<!--1.0 means on pt sample per pixel. A higher value gives less noise-->
<ptAccuracy>1</ptAccuracy>
<!--World space value. A small value makes the cache more detailed-->
<ptPointSize>0</ptPointSize>
<!--Cache direct light as well, boosts speed-->
<ptCacheDirectLight>true</ptCacheDirectLight>
```

```
  <!--Check visibility before using values in the cache-->
  <ptCheckVisibility>false</ptCheckVisibility>
</GISettings>
```

Appendix D

Copyrights

D.1 Boost

The Beast API uses Boost C++ Libraries. This is the license:

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

D.2 Google C++ Testing Framework

The Beast API uses Google C++ Testing Framework. This is the license:

```
Copyright 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

    * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
    * Neither the name of Google Inc. nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

D.3 HMAC SHA1

The Beast API uses HMAC SHA1 Library. This is the license:

```
hmac_sha1.h

Version 1.0.0

Written by Aaron D. Gifford <me@aarongifford.com>

Copyright 1998, 2000 Aaron D. Gifford.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR(S) AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR(S) OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

D.4 libpng

The Beast API uses libpng. This is the license:

```
This copy of the libpng notices is provided for your convenience.  In case of
any discrepancy between this copy and the notices in the file png.h that is
included in the libpng distribution, the latter shall prevail.

COPYRIGHT NOTICE, DISCLAIMER, and LICENSE:

If you modify libpng you may insert additional notices immediately following
this sentence.

libpng versions 1.2.6, August 15, 2004, through 1.2.35, February 14, 2009, are
Copyright (c) 2004, 2006-2008 Glenn Randers-Pehrson, and are
distributed according to the same disclaimer and license as libpng-1.2.5
with the following individual added to the list of Contributing Authors

   Cosmin Truta

libpng versions 1.0.7, July 1, 2000, through 1.2.5 - October 3, 2002, are
Copyright (c) 2000-2002 Glenn Randers-Pehrson, and are
distributed according to the same disclaimer and license as libpng-1.0.6
with the following individuals added to the list of Contributing Authors

   Simon-Pierre Cadieux
   Eric S. Raymond
   Gilles Vollant

and with the following additions to the disclaimer:
```

There is no warranty against interference with your enjoyment of the library or against infringement. There is no warranty that our efforts or the library will fulfill any of your particular purposes or needs. This library is provided with all faults, and the entire risk of satisfactory quality, performance, accuracy, and effort is with the user.

libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are Copyright (c) 1998, 1999 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-0.96, with the following individuals added to the list of Contributing Authors:

Tom Lane
Glenn Randers-Pehrson
Willem van Schaik

libpng versions 0.89, June 1996, through 0.96, May 1997, are Copyright (c) 1996, 1997 Andreas Dilger
Distributed according to the same disclaimer and license as libpng-0.88, with the following individuals added to the list of Contributing Authors:

John Bowler
Kevin Bracey
Sam Bushell
Magnus Holmgren
Greg Roelofs
Tom Tanner

libpng versions 0.5, May 1995, through 0.88, January 1996, are Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.

For the purposes of this copyright and license, "Contributing Authors" is defined as the following set of individuals:

Andreas Dilger
Dave Martindale
Guy Eric Schalnat
Paul Schmidt
Tim Wegner

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage.

Permission is hereby granted to use, copy, modify, and distribute this source code, or portions hereof, for any purpose, without fee, subject to the following restrictions:

1. The origin of this source code must not be misrepresented.

2. Altered versions must be plainly marked as such and must not be misrepresented as being the original source.

3. This Copyright notice may not be removed or altered from any source or altered source distribution.

The Contributing Authors and Group 42, Inc. specifically permit, without fee, and encourage the use of this source code as a component to supporting the PNG file format in commercial products. If you use this source code in a product, acknowledgment is not required but would be appreciated.

A "png_get_copyright" function is available, for convenient use in "about" boxes and the like:

```
   printf("%s",png_get_copyright(NULL));
```

Also, the PNG logo (in PNG format, of course) is supplied in the files "pngbar.png" and "pngbar.jpg (88x31) and "pngnow.png" (98x31).

Libpng is OSI Certified Open Source Software. OSI Certified Open Source is a certification mark of the Open Source Initiative.

Glenn Randers-Pehrson
glennrp at users.sourceforge.net
February 14, 2009

D.5 OpenEXR

The Beast API uses OpenEXR. This is the license:

Copyright (c) 2004, Industrial Light & Magic, a division of Lucas Digital Ltd. LLC

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
*   Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
*   Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer

in the documentation and/or other materials provided with the distribution.

* Neither the name of Industrial Light & Magic nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

D.6 TinyXML

The Beast API uses TinyXML. This is the license:

TinyXML is released under the zlib license:

This software is provided ’as-is’, without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

D.7 utf8-cpp

The Beast API uses utf8-cpp library. This is the license:

Copyright 2006 Nemanja Trifunovic

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

D.8 zlib

The Beast API uses zlib. This is the license:

version 1.2.3, July 18th, 2005

Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

```
Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu


The data format used by the zlib library is described by RFCs (Request for
Comments) 1950 to 1952 in the files http://www.ietf.org/rfc/rfc1950.txt
(zlib format), rfc1951.txt (deflate format) and rfc1952.txt (gzip format).
```

Bibliography

[1] Tomas Akenine-Möller and Eric Haines and Natty Hoffman, Real-Time Rendering. A. K. Peters, Ltd., 3rd Edition, 2008.

[2] Steve Smith, Picture Perfect: Gamma through the Rendering Pipeline. Gamefest 2007 Presentation, http://www.xnagamefest.com/presentations07.htm.

[3] Charles Poynton, Frequently-Asked Questions about Gamma. http://www.poynton.com/GammaFAQ.html.

Part II

API Reference

Chapter 4

Data Structure Documentation

4.1 ILBCameraHandle Struct Reference

```
#include <beastapitypes.h>
```

4.1.1 Detailed Description

Handle for Beast cameras

Intentionally hidden implementation

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

- beastapitypes.h

4.2 ILBFramebufferHandle Struct Reference

```
#include <beastapitypes.h>
```

4.2.1 Detailed Description

Handle for Beast framebuffers Intentionally hidden implementation

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

- beastapitypes.h

4.3 ILBInstanceHandle Struct Reference

```
#include <beastapitypes.h>
```

4.3.1 Detailed Description

Handle for Beast instance

Intentionally hidden implementation

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

- beastapitypes.h

4.4 ILBJobHandle Struct Reference

```
#include <beastapitypes.h>
```

4.4.1 Detailed Description

Handle for Beast jobs

Intentionally hidden implementation

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

- beastapitypes.h

4.5 ILBLightHandle Struct Reference

```
#include <beastapitypes.h>
```

4.5.1 Detailed Description

Handle for Beast light sources

Intentionally hidden implementation

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

- beastapitypes.h

4.6 ILBLightPassEntryHandle Struct Reference

```
#include <beastapitypes.h>
```

4.6.1 Detailed Description

Handle for Light pass entry

Intentionally hidden implementation

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

- beastapitypes.h

4.7 ILBLinearRGB Struct Reference

```
#include <beastapitypes.h>
```

Data Fields

- float r

  Red component.

- float g

  Green component.

- float b

  Blue component.

4.7.1 Detailed Description

Color definition All colors are expressed in linear space as opposed to gamma corrected.

High dynamic range supported.

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

- beastapitypes.h

4.8 ILBLinearRGBA Struct Reference

```
#include <beastapitypes.h>
```

Data Fields

- float r

  Red component.

- float g

  Green component.

- float b

  Blue component.

- float a

  Alpha.

4.8.1 Detailed Description

Color with alpha definition All colors are expressed in linear space as opposed to gamma corrected.

Alpha values must be between 0 and 1.0.

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

- beastapitypes.h

4.9 ILBManagerHandle Struct Reference

```
#include <beastapitypes.h>
```

4.9.1 Detailed Description

Handle for Beast managers

Intentionally hidden implementation

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

- beastapitypes.h

4.10 ILBMaterialHandle Struct Reference

```
#include <beastapitypes.h>
```

4.10.1 Detailed Description

Handle for Beast materials

Intentionally hidden implementation

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

- beastapitypes.h

4.11 ILBMatrix4x4 Struct Reference

```
#include <beastapitypes.h>
```

Data Fields

- float m [16]

4.11.1 Detailed Description

Matrix for transformations.

The data is stored line by line (the opposite of OpenGL)

Indices are laid out like this:

```
(00 01 02 03)

(04 05 06 07)

(08 09 10 11)

(12 13 14 15)
```

4.11.2 Field Documentation

4.11.2.1 float ILBMatrix4x4::m[16]

The actual data in the matrix

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

- beastapitypes.h

4.12 ILBMeshHandle Struct Reference

```
#include <beastapitypes.h>
```

4.12.1 Detailed Description

Handle for Beast meshes

Intentionally hidden implementation

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

- beastapitypes.h

4.13 ILBPointCloudHandle Struct Reference

```
#include <beastapitypes.h>
```

4.13.1 Detailed Description

Handle for Beast point clouds

Intentionally hidden implementation

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

- beastapitypes.h

4.14 ILBRenderPassHandle Struct Reference

```
#include <beastapitypes.h>
```

4.14.1 Detailed Description

Handle for Beast render pass

Intentionally hidden implementation

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

- beastapitypes.h

4.15 ILBSceneHandle Struct Reference

```
#include <beastapitypes.h>
```

4.15.1 Detailed Description

Handle for Beast scenes

Intentionally hidden implementation

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

- beastapitypes.h

4.16 ILBStringHandle Struct Reference

```
#include <beastapitypes.h>
```

4.16.1 Detailed Description

Handle for Beast strings

Intentionally hidden implementation

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

- beastapitypes.h

4.17 ILBTargetEntity Struct Reference

```
#include <beastapitypes.h>
```

4.17.1 Detailed Description

Handle for Beast target entities Intentionally hidden implementation

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

- beastapitypes.h

4.18 ILBTargetHandle Struct Reference

```
#include <beastapitypes.h>
```

4.18.1 Detailed Description

Handle for Beast target specification

Intentionally hidden implementation

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

- beastapitypes.h

4.19 ILBTextureHandle Struct Reference

```
#include <beastapitypes.h>
```

4.19.1 Detailed Description

Handle for Beast textures

Intentionally hidden implementation

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

- beastapitypes.h

4.20 ILBVec2 Struct Reference

```
#include <beastapitypes.h>
```

Data Fields

- float x

  x

- float y

  y

4.20.1 Detailed Description

Two dimensional geometric vector type.

Used for both points and vectors

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

- beastapitypes.h

4.21 ILBVec3 Struct Reference

```
#include <beastapitypes.h>
```

Data Fields

- float x

  x

- float y

  y

- float z

  z

4.21.1 Detailed Description

Three dimensional geometric vector type.

Used for both points and vectors, what is what should be obvious from the context.

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

- beastapitypes.h

Chapter 5

File Documentation

5.1 beastapitypes.h File Reference

Data Structures

- struct ILBVec2
- struct ILBVec3
- struct ILBLinearRGB
- struct ILBLinearRGBA
- struct ILBMatrix4x4

Defines

- #define ILB_STRING_ENCODING ILB_SE_UTF8
- #define ILB_DLL_FUNCTION

Typedefs

- typedef char ILBChar8
- typedef unsigned short ILBChar16
- typedef unsigned int ILBChar32
- typedef ILBChar8 ILBCharType
- typedef ILBCharType * ILBString
- typedef const ILBCharType * ILBConstString
- typedef int32 ILBBool

Enumerations

- enum ILBStringEncoding { ILB_SE_ANSI, ILB_SE_UTF8, ILB_SE_UTF16, ILB_SE_UTF32 }

- enum ILBStatus {
  ILB_ST_SUCCESS = 0, ILB_ST_INVALID_PARAMETER, ILB_ST_MEMORY_ALLOC_ERROR, ILB_ST_DUPLICATE_NAME_ERROR,
  ILB_ST_FUNCTION_NOT_IMPLEMENTED, ILB_ST_INVALID_OBJECT_STATE, ILB_ST_INVALID_HANDLE, ILB_ST_FILE_IO_ERROR,
  ILB_ST_UNKNOWN_OBJECT, ILB_ST_NOT_SUPPORTED, ILB_ST_UNHANDLED_EXCEPTION, ILB_ST_JOB_EXECUTION_FAILURE,
  ILB_ST_ATLAS_EXECUTION_FAILURE, ILB_ST_LAST_ERROR }
- enum ILBLightLinkMode { ILB_LL_EXCLUDING = 0, ILB_LL_INCLUDING }

5.1.1 Detailed Description

This header is the base for getting platform consistent types for the Beast API

5.1.2 Define Documentation

5.1.2.1 #define ILB_DLL_FUNCTION

Setup for getting functions exported or imported from the DLL on windows

5.1.2.2 #define ILB_STRING_ENCODING ILB_SE_UTF8

Defined to be the string encoding that is set. Don't set directly, use one of the following defines:

ILB_STRING_UTF8

ILB_STRING_UTF16

ILB_STRING_UTF32

ILB_STRING_ANSI

5.1.3 Typedef Documentation

5.1.3.1 typedef int32 ILBBool

Bool type with a well defined size to avoid compatibility-problems.

5.1.3.2 typedef unsigned short ILBChar16

Character type for 16 bit strings

5.1.3.3 typedef unsigned int ILBChar32

Character type for 32 bit strings. May need special treatment for native utf32 platforms (i.e wchar_t instead of int32)

5.1.3.4 typedef char ILBChar8

Character type for 8 bit strings

5.1.3.5 ILBCharType

Will be typedef:ed to the proper character type depending on the selected string encoding

5.1.3.6 typedef const ILBCharType* ILBConstString

Beast api const string type. Represents different things depending on the selected string encoding to make sure it's compatible with string constants.

5.1.3.7 typedef ILBCharType* ILBString

Beast api string type. Represents different things depending on the selected string encoding to make sure it's compatible with string constants.

5.1.4 Enumeration Type Documentation

5.1.4.1 enum ILBLightLinkMode

Enum defining if a light link is inclusive or exclusive

Enumerator:

ILB_LL_EXCLUDING Excludes the supplied lights/objects

ILB_LL_INCLUDING Includes the supplied lights/objects

5.1.4.2 enum ILBStatus

Status codes for Beast API calls

Enumerator:

ILB_ST_SUCCESS The call was successfully completed!

ILB_ST_INVALID_PARAMETER One or more parameters were not in valid range or in some other way not valid for this call.

ILB_ST_MEMORY_ALLOC_ERROR Beast failed to allocate memory somewhere down the line of this call

ILB_ST_DUPLICATE_NAME_ERROR An object with the same name already existed

ILB_ST_FUNCTION_NOT_IMPLEMENTED This function is not implemented yet. Should only happen internally

ILB_ST_INVALID_OBJECT_STATE The object the function was called on was in a state where the function isn't valid to call.

ILB_ST_INVALID_HANDLE The object handle used is not valid.

ILB_ST_FILE_IO_ERROR There was some kind of file problem (invalid filename, permission etc).

ILB_ST_UNKNOWN_OBJECT A handle to an unknown object was requested

ILB_ST_NOT_SUPPORTED The requested functionality is not supported in the current configuration

ILB_ST_UNHANDLED_EXCEPTION The api generated an exception we didn't expect.

ILB_ST_JOB_EXECUTION_FAILURE An external tool returned an error

ILB_ST_ATLAS_EXECUTION_FAILURE Atlasing failed.

ILB_ST_LAST_ERROR Dummy entry to be able to loop over all errors

5.1.4.3 enum ILBStringEncoding

Defines the different supported string encodings

Enumerator:

ILB_SE_ANSI Ansi encoding, the one used by default for windows source files

ILB_SE_UTF8 UTF-8 encoding

ILB_SE_UTF16 UTF-16 encoding

ILB_SE_UTF32 UTF-32 encoding.
Currently not supported

5.2 beastcamera.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBEnvironmentCameraType { ILB_ECT_CUBEMAP = 0, ILB_ECT_BALL, ILB_ECT_LATLONG }

Functions

- ILBStatus ILBCreatePerspectiveCamera (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 *transform, ILBCameraHandle *camera)
- ILBStatus ILBCreateEnvironmentCamera (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 *transform, ILBEnvironmentCameraType type, ILBCameraHandle *camera)
- ILBStatus ILBSetFov (ILBCameraHandle camera, float horizontalFovRadians, float pixelAspectRatio)

5.2.1 Detailed Description

The beast camera function definitions

5.2.2 Enumeration Type Documentation

5.2.2.1 enum ILBEnvironmentCameraType

Environment camera types

Enumerator:

ILB_ECT_CUBEMAP Cubemap environment camera.

ILB_ECT_BALL Ball environment camera.

ILB_ECT_LATLONG Latlong environment camera.

5.2.3 Function Documentation

5.2.3.1 ILBStatus ILBCreateEnvironmentCamera (ILBSceneHandle *scene*, ILBConstString *name*, const ILBMatrix4x4 * *transform*, ILBEnvironmentCameraType *type*, ILBCameraHandle * *camera*)

Add an environment camera to the scene.

Parameters:

scene the scene the camera should be a part of

name the name of the camera, must be unique within the scene.

transform the object space to world space transform for this camera

type the type of environment camera, i.e Ball, Cubemap or Latlong

camera the handle to store the generated camera in

Returns:

The result of the operation.

5.2.3.2 ILBStatus **ILBCreatePerspectiveCamera** (ILBSceneHandle ***scene***, ILBConstString ***name***, **const** ILBMatrix4x4 * ***transform***, ILBCameraHandle * ***camera***)

Add a camera to the scene. The camera looks in the negative z direction, positive x in camera space maps to right in screen space. positive y maps to up i screen space.

Parameters:

scene the scene the camera should be a part of

name the name of the camera, must be unique within the scene.

transform the object space to world space transform for this camera

camera the handle to store the generated camera in

Returns:

The result of the operation.

5.2.3.3 ILBStatus **ILBSetFov** (ILBCameraHandle ***camera***, **float** ***horizontalFovRadians***, **float** ***pixelAspectRatio***)

Sets the fov of the camera. Only works on perspective cameras. The vertical fov will be generated from the horizontal fov and the image dimensions

Parameters:

camera the camera to set fov for

horizontalFovRadians the field of view in the X direction in radians.

Note that it refers to the complete field of vision, not the angle to the forward direction. Negative horizontalFovRadians is currently not supported

pixelAspectRatio the aspect ratio of a pixel, expressed as the x / y

5.3 beastframebuffer.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBChannelSelection {
  ILB_CS_R = 0, **ILB_CS_G**, **ILB_CS_B**, **ILB_CS_A**,
  ILB_CS_Z, **ILB_CS_RGB**, **ILB_CS_RGBA**, **ILB_CS_RGBAZ**,
  ILB_CS_ALL }

Functions

- ILBStatus ILBGetChannelCount (ILBFramebufferHandle fb, int32 ∗channels)
- ILBStatus ILBGetChannelName (ILBFramebufferHandle fb, int32 index, ILBStringHandle ∗name)
- ILBStatus ILBGetResolution (ILBFramebufferHandle fb, int32 ∗width, int32 ∗height)
- ILBStatus ILBReadRegionHDR (ILBFramebufferHandle fb, int32 minX, int32 minY, int32 maxX, int32 maxY, ILBChannelSelection channels, float ∗target)
- ILBStatus ILBReadRegionLDR (ILBFramebufferHandle fb, int32 minX, int32 minY, int32 maxX, int32 maxY, ILBChannelSelection channels, float gamma, unsigned char ∗target)
- ILBStatus ILBDestroyFramebuffer (ILBFramebufferHandle fb)

5.3.1 Detailed Description

Copyright (c) 2008-2009 Illuminate Labs AB.

You may use this code for any project covered by a separate written license agreement with Illuminate Labs AB. You may also use this code as part of an evaluation of Illuminate Labs AB's software.

You bear the entire risk of using this code. The code is provided as-is and Illuminate Labs AB gives no guarantees or warranties whatsoever. Illuminate Labs AB excludes the implied warranties of merchantability, fitness for a particular purpose and non-infringement. The Beast framebuffer functions

5.3.2 Enumeration Type Documentation

5.3.2.1 enum ILBChannelSelection

Default selections of channels when reading out pixel/vertex data

5.3.3 Function Documentation

5.3.3.1 ILBStatus **ILBDestroyFramebuffer** (ILBFramebufferHandle ***fb***)

Destroys and frees all memory related to a framebuffer. The framebuffer handle will be invalid afterwards.

Note this will happen automatically when the job is destroyed. Use this function to avoid using more temporary memory than necessary when importing results.

Parameters:

fb the framebuffer to erase

Returns:

The result of the operation.

5.3.3.2 ILBStatus **ILBGetChannelCount** (ILBFramebufferHandle ***fb***, **int32** ∗ ***channels***)

Gets the number of channels in the framebuffer

Parameters:

fb the framebuffer to get the channel count for

channels pointer to the variable to receive the number of channels

Returns:

The result of the operation.

5.3.3.3 ILBStatus **ILBGetChannelName** (ILBFramebufferHandle ***fb***, **int32** ***index***, ILBStringHandle ∗ ***name***)

Gets the name for a channel

Parameters:

fb the framebuffer to get the channel from

index the index of the channel to get

name pointer to the string handle to receive the channel name

Returns:

The result of the operation.

5.3.3.4 ILBStatus **ILBGetResolution** (ILBFramebufferHandle ***fb***, **int32** ∗ ***width***, **int32** ∗ ***height***)

Gets the resolution for a framebuffer

Parameters:

fb the framebuffer to get the resolution of

width pointer to where the width shall be written

height pointer to where the height shall be written

Returns:

The result of the operation.

5.3.3.5 ILBStatus **ILBReadRegionHDR** (ILBFramebufferHandle ***fb***, **int32** ***minX***, **int32** ***minY***, **int32** ***maxX***, **int32** ***maxY***, ILBChannelSelection ***channels***, **float** ∗ ***target***)

Reads back a region in a framebuffer. The region is specified as inclusive for the lower part and exclusive for the higher part. I.E to read a 512x512 buffer use 0, 0, 512, 512

Parameters:

fb the framebuffer to read from

minX the left limit of the region

minY the lower limit of the region

maxX the right limit of the region

maxY the upper limit of the region

channels what channels to read out from the buffer

target the buffer to write the result to. Must be (maxY - minY) ∗ (maxX - minX) ∗ channelCount floats big

Returns:

The result of the operation.

5.3.3.6 ILBStatus **ILBReadRegionLDR** (ILBFramebufferHandle ***fb***, **int32** ***minX***, **int32** ***minY***, **int32** ***maxX***, **int32** ***maxY***, ILBChannelSelection ***channels***, **float** ***gamma***, **unsigned char** ∗ ***target***)

Reads back a region in a framebuffer. The region is specified as inclusive for the lower part and exclusive for the higher part. I.E to read a 512x512 buffer use 0, 0, 512, 512

Parameters:

fb the framebuffer to read from

minX the left limit of the region

minY the lower limit of the region

maxX the right limit of the region

maxY the upper limit of the region

channels what channels to read out from the buffer

gamma the gamma space to encode the image data in

target the buffer to write the result to. Must be (maxY - minY) $*$ (maxX - minX) $*$ channelCount bytes big

Returns:

The result of the operation.

5.4 beastinstance.h File Reference

```
#include "beastapitypes.h"
```

Typedefs

- typedef uint32 ILBRenderStatsMask

Enumerations

- enum ILBRenderStats {

  ILB_RS_PRIMARY_VISIBILITY = 0x00000001, ILB_RS_CAST_SHADOWS = 0x00000002, ILB_RS_RECEIVE_SHADOWS = 0x00000004, ILB_RS_VISIBLE_IN_REFLECTIONS = 0x00000020,

  ILB_RS_VISIBLE_IN_REFRACTIONS = 0x00000040, ILB_RS_VISIBLE_IN_FINAL_GATHER = 0x00000080, ILB_RS_DOUBLE_SIDED = 0x00000100, ILB_RS_OPPOSITE = 0x00000200,

  ILB_RS_CAST_GI = 0x00000400, ILB_RS_RECEIVE_GI = 0x00000800, ILB_RS_CAST_OCCLUSION = 0x00004000, ILB_RS_RECEIVE_OCCLUSION = 0x00008000,

  ILB_RS_SELF_OCCLUSION = 0x00040000, ILB_RS_SHADOW_BIAS = 0x00080000 }
- enum ILBRenderStatOperation { ILB_RSOP_ENABLE, ILB_RSOP_DISABLE }

Functions

- ILBStatus ILBCreateInstance (ILBSceneHandle scene, ILBMeshHandle mesh, ILBConstString name, const ILBMatrix4x4 ∗transform, ILBInstanceHandle ∗instance)
- ILBStatus ILBSetRenderStats (ILBInstanceHandle instance, ILBRenderStatsMask stats, ILBRenderStatOperation operation)
- ILBStatus ILBAddInstanceLightLinks (ILBInstanceHandle instance, ILBLightLinkMode mode, const ILBLightHandle ∗lightSources, int32 count)
- ILBStatus ILBSetMaterialOverrides (ILBInstanceHandle instance, ILBMaterialHandle ∗materials, int32 materialCount)
- ILBStatus ILBAddLODInstance (ILBInstanceHandle lowRes, ILBInstanceHandle highRes)

5.4.1 Detailed Description

The api for specifying instances of meshes in Beast

5.4.2 Typedef Documentation

5.4.2.1 typedef uint32 ILBRenderStatsMask

Type representing multiple renderstats. Combine renderstats using the or operator (|).

Example:

ILBRenderStatsMask rsMask = ILB_RS_SELF_OCCLUSION | ILB_RS_PRIMARY_-VISIBILITY;

5.4.3 Enumeration Type Documentation

5.4.3.1 enum ILBRenderStatOperation

Selects if the render stats should be enabled or disabled

Enumerator:

ILB_RSOP_ENABLE Sets the render stats supplied to true

ILB_RSOP_DISABLE Sets the render stats supplied to false

5.4.3.2 enum ILBRenderStats

Beast render stats

Enumerator:

ILB_RS_PRIMARY_VISIBILITY Controls whether the object should be visible for primary rays All other effects such as shadow casting and occlusion casting is unaffected.

Default: Enabled

ILB_RS_CAST_SHADOWS Controls whether the object should cast shadows.

Default: Enabled

ILB_RS_RECEIVE_SHADOWS Controls whether the object should receive shadows.

Default: Enabled

ILB_RS_VISIBLE_IN_REFLECTIONS Controls whether the object should be visible for reflection rays

Default: Enabled

ILB_RS_VISIBLE_IN_REFRACTIONS Controls whether the object should be visible for refraction rays

Default: Enabled

ILB_RS_VISIBLE_IN_FINAL_GATHER Controls whether the object should be visible for final gather rays

Default: Enabled

ILB_RS_DOUBLE_SIDED Controls whether the object should be single sided or double sided. If single sided, only polygons that are defined CCW from the ray are visible.

Default: Enabled

ILB_RS_OPPOSITE Only applies when single sided. It flips the test for rejecting single sided polygons.

Default: Disabled

ILB_RS_CAST_GI If this is set to false, the object is black for GI purposes. It's still in the scene and casts occlusion, but no light bounces off it and it casts no color bleeding on neighbor objects.

Default: Enabled

ILB_RS_RECEIVE_GI Controls whether the object is receiving any GI. If disabled it only gets direct light.

Default: Enabled

ILB_RS_CAST_OCCLUSION Controls whether the object is an occluder when rendering ambient occlusion. Default: Enabled

ILB_RS_RECEIVE_OCCLUSION Controls whether the object is receiving any occlusion when rendering ambient occlusion.

Default: Enabled

ILB_RS_SELF_OCCLUSION Controls whether the object cast shadows on itself.

Default: Enabled

ILB_RS_SHADOW_BIAS If enabled shadow rays will be biased in order to get smooth shadows on smoothed meshes.

Default: Disabled

5.4.4 Function Documentation

5.4.4.1 ILBStatus ILBAddInstanceLightLinks (ILBInstanceHandle *instance*, ILBLightLinkMode *mode*, const ILBLightHandle ∗ *lightSources*, int32 *count*)

Adds an object centric light link list

Parameters:

instance the instance to add light links to

mode sets whether the light links are inclusive or exclusive

lightSources an array of light sources that should be linked

count the number of light sources present in the lightSources array

Returns:

The result of the operation.

5.4.4.2 ILBStatus **ILBAddLODInstance** (ILBInstanceHandle ***lowRes***, ILBInstanceHandle ***highRes***)

Connects two meshes so one acts as a high resolution instances and one as a lod. This is used for normal map generation etc. When connected, the low resolution mesh will automatically be set as invisible using renderstats. There can be many high resolution instances for every low resolution instance. An instance can act as high resolution mesh for many low resolution meshes

Parameters:

lowRes the instance to act as low resolution mesh in this lod relationship. Will be hidden by this call.

highRes the high resolution mesh in this lod relationship

Returns:

The result of the operation.

5.4.4.3 ILBStatus **ILBCreateInstance** (ILBSceneHandle ***scene***, ILBMeshHandle ***mesh***, ILBConstString ***name***, **const** ILBMatrix4x4 ∗ ***transform***, ILBInstanceHandle ∗ ***instance***)

Add an instance to the scene.

Parameters:

scene the scene the instance should be a part of

mesh the mesh the instance should reference

name the name of the instance, must be unique within the scene.

transform the object space to world space transform for this mesh

instance the handle to store the generated instance in

Returns:

The result of the operation.

5.4.4.4 ILBStatus **ILBSetMaterialOverrides** (ILBInstanceHandle ***instance***, ILBMaterialHandle ∗ ***materials***, **int32** ***materialCount***)

Overrides materials for this instance.

Parameters:

instance the instance to override materials for

materials an array of materials to override the one specified in the mesh with. It should be in the same order as the material groups in the mesh. A 0 pointer will keep the default material. If the material lists size doesn't match, unassigned materials will stay default and overflowing materials will be ignored.

materialCount the number of override materials available in the materials array.

Returns:

The result of the operation.

5.4.4.5 ILBStatus **ILBSetRenderStats** (ILBInstanceHandle ***instance***, ILBRenderStatsMask ***stats***, ILBRenderStatOperation ***operation***)

Sets render stats on an instance.

Parameters:

instance the instance to set the render stats on

stats the stats to modify. Can be multiple render stats or:ed together

operation selects whether to enable or disable the selected render stats.

Returns:

The result of the operation.

5.5 beastjob.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBJobStatus {
  ILB_JS_SUCCESS = 0, ILB_JS_CANCELLED, ILB_JS_INVALID_LICENSE, ILB_JS_-CMDLINE_ERROR,
  ILB_JS_CONFIG_ERROR, ILB_JS_CRASH, ILB_JS_OTHER_ERROR = 0x10000001 }
- enum ILBShowResults { ILB_SR_NO_DISPLAY = 0, ILB_SR_CLOSE_WHEN_DONE, ILB_SR_KEEP_OPEN }
- enum ILBDistributionType { ILB_RD_FORCE_LOCAL = 0, ILB_RD_AUTODETECT, ILB_RD_FORCE_DISTRIBUTED }

Functions

- ILBStatus ILBCreateJob (ILBManagerHandle beastManager, ILBConstString uniqueName, ILBSceneHandle scene, ILBConstString jobXML, ILBJobHandle ∗job)
- ILBStatus ILBDestroyJob (ILBJobHandle job)
- ILBStatus ILBSetJobOutputPath (ILBJobHandle job, ILBConstString path)
- ILBStatus ILBStartJob (ILBJobHandle job, ILBShowResults showResults, ILBDistributionType distribution)
- ILBStatus ILBWaitJobDone (ILBJobHandle job, int32 timeout)
- ILBStatus ILBIsJobRunning (ILBJobHandle job, ILBBool ∗result)
- ILBStatus ILBIsJobCompleted (ILBJobHandle job, ILBBool ∗result)
- ILBStatus ILBGetJobResult (ILBJobHandle job, ILBJobStatus ∗status)
- ILBStatus ILBCancelJob (ILBJobHandle job)
- ILBStatus ILBGetJobProgress (ILBJobHandle job, ILBStringHandle ∗jobName, int32 ∗progress)
- ILBStatus ILBJobHasNewProgress (ILBJobHandle job, ILBBool ∗newActivity, ILBBool ∗newProgress)
- ILBStatus ILBExecuteBeast (ILBManagerHandle bm, ILBJobHandle job, ILBShowResults showResults, ILBDistributionType distribution, ILBJobStatus ∗status)

5.5.1 Detailed Description

The beast job function definitions

5.5.2 Enumeration Type Documentation

5.5.2.1 enum ILBDistributionType

Sets how beast should render distributed

Enumerator:

ILB_RD_FORCE_LOCAL Force a local render

ILB_RD_AUTODETECT Render distributed if possible, otherwise fallback on local rendering

ILB_RD_FORCE_DISTRIBUTED Force a distributed render, fails if distribution is not available

5.5.2.2 enum ILBJobStatus

Status codes for Beast API calls

Enumerator:

ILB_JS_SUCCESS This was a triumph! I'm making a note here; Huge Success!

ILB_JS_CANCELLED Job was aborted by external means.

ILB_JS_INVALID_LICENSE Beast does not have a valid license.

ILB_JS_CMDLINE_ERROR Error parsing the command line

ILB_JS_CONFIG_ERROR Error parsing the config files

ILB_JS_CRASH Beast crashed, sorry.

ILB_JS_OTHER_ERROR Other Error

5.5.2.3 enum ILBShowResults

Sets how to handle the Beast window when rendering

Enumerator:

ILB_SR_NO_DISPLAY Don't display the render window

ILB_SR_CLOSE_WHEN_DONE Show the render window and close it when the rendering is done.

ILB_SR_KEEP_OPEN Show the render window and keep it open until the user closes it or the job is destroyed.

5.5.3 Function Documentation

5.5.3.1 ILBStatus ILBCancelJob (ILBJobHandle *job*)

Cancels a running job

Parameters:

job the job to cancel

5.5.3.2 ILBStatus **ILBCreateJob** (ILBManagerHandle ***beastManager***, ILBConstString ***uniqueName***, ILBSceneHandle ***scene***, ILBConstString ***jobXML***, ILBJobHandle $*$ ***job***)

Parameters:

beastManager the beast manager to create the job for

uniqueName a unique name for the job

scene the scene to render

jobXML the config XML file to use. Use empty string for default rendering (no GI or supersampling)

job pointer to where the job handle should be stored

5.5.3.3 ILBStatus **ILBDestroyJob** (ILBJobHandle ***job***)

Destroys a job

Parameters:

job the job to destroy

5.5.3.4 ILBStatus **ILBExecuteBeast** (ILBManagerHandle ***bm***, ILBJobHandle ***job***, ILBShowResults ***showResults***, ILBDistributionType ***distribution***, ILBJobStatus $*$ ***status***)

Convenience function to execute a Beast job. Blocks until the job is done or fails

Parameters:

bm the beast manager to use

job the job to execute

showResults sets how the beast window should be handled

distribution Sets how to distribute the rendering

status the result of the rendering

5.5.3.5 ILBStatus **ILBGetJobProgress** (ILBJobHandle ***job***, ILBStringHandle $*$ ***jobName***, **int32** $*$ ***progress***)

Gets the current status of a job.

Parameters:

job the job to get progress for

jobName pointer to a string object that receives the name of job being executed Set to 0 to ignore this parameter.

progress to the completion percentage of the current activity

5.5.3.6 ILBStatus **ILBGetJobResult** (ILBJobHandle ***job***, ILBJobStatus ∗ ***status***)

Returns the result of the job as a JobStatus

Parameters:

job the job to get the result for

status pointer to the JobStatus

5.5.3.7 ILBStatus **ILBIsJobCompleted** (ILBJobHandle ***job***, ILBBool ∗ ***result***)

Checks if the job is completed. Note that a running job can be completed if the user has selected to keep the render window open. A job that is not running might not have finished if it was aborted or had errors.

Parameters:

job the job to check

result set to true if the job is completed, false otherwise

5.5.3.8 ILBStatus **ILBIsJobRunning** (ILBJobHandle ***job***, ILBBool ∗ ***result***)

Checks if the job is running.

Parameters:

job The job to check

result Is set to true if job is running, false otherwise

5.5.3.9 ILBStatus **ILBJobHasNewProgress** (ILBJobHandle ***job***, ILBBool ∗ ***newActivity***, ILBBool ∗ ***newProgress***)

Checks if the progress of a Job has been updated since the last time ILBGetJobProgress was called.

Parameters:

job The job to check if it has progress

newActivity set to true if a new activity has started

newProgress set to true if the progress has been updated

5.5.3.10 ILBStatus **ILBSetJobOutputPath** (ILBJobHandle ***job***, ILBConstString ***path***)

Sets the output directory fpr the job. If this function is not called output files will end up in the cache hierarchy.

Parameters:

job the job to set directory for

path the path to the output directory

5.5.3.11 ILBStatus **ILBStartJob** (ILBJobHandle ***job***, ILBShowResults ***showResults***, ILBDistributionType ***distribution***)

Starts a job

Parameters:

job the job to start

showResults Specifies the behaviour of the render window

distribution Sets how to distribute the rendering

5.5.3.12 ILBStatus **ILBWaitJobDone** (ILBJobHandle ***job***, int32 ***timeout***)

Waits until a job is done or until there is progress updates

Parameters:

job The job to wait for

timeout The maximum time to wait in milliseconds

5.6 beastlightsource.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBFalloffType { ILB_FO_EXPONENT = 0, ILB_FO_MAX_RANGE }

Functions

- ILBStatus ILBCreatePointLight (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 ∗transform, const ILBLinearRGB ∗intensity, ILBLightHandle ∗light)
- ILBStatus ILBCreateSpotLight (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 ∗transform, const ILBLinearRGB ∗intensity, ILBLightHandle ∗light)
- ILBStatus ILBCreateDirectionalLight (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 ∗transform, const ILBLinearRGB ∗intensity, ILBLightHandle ∗light)
- ILBStatus ILBCreateAreaLight (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 ∗transform, const ILBLinearRGB ∗intensity, ILBLightHandle ∗light)
- ILBStatus ILBCreateWindowLight (ILBSceneHandle scene, ILBConstString name, const ILBMatrix4x4 ∗transform, const ILBLinearRGB ∗intensity, ILBLightHandle ∗light)
- ILBStatus ILBSetCastShadows (ILBLightHandle light, ILBBool castShadows)
- ILBStatus ILBSetShadowRadius (ILBLightHandle light, float radius)
- ILBStatus ILBSetShadowAngle (ILBLightHandle light, float angleRadians)
- ILBStatus ILBSetShadowSamples (ILBLightHandle light, int32 samples)
- ILBStatus ILBAddLightLightLinks (ILBLightHandle light, ILBLightLinkMode mode, const ILBInstanceHandle ∗instances, int32 count)
- ILBStatus ILBSetFalloff (ILBLightHandle light, ILBFalloffType type, float exponent, float cutoff, ILBBool clampToOne)
- ILBStatus ILBSetSpotlightCone (ILBLightHandle light, float angleRadians, float penumbraAngleRadians, float penumbraExponent)
- ILBStatus ILBSetIntensityScale (ILBLightHandle light, float directIntensity, float indirectIntensity)
- ILBStatus ILBSetLightRampEntry (ILBLightHandle light, float position, const ILBLinearRGB ∗value)
- ILBStatus ILBSetLightProjectedTexture (ILBLightHandle light, ILBTextureHandle texture)

5.6.1 Detailed Description

The api for specifying light sources in beast

5.6.2 Enumeration Type Documentation

5.6.2.1 enum ILBFalloffType

Describes different falloff for light sources

Enumerator:

ILB_FO_EXPONENT Computes falloff as `(1.0f / distance) ^ exponent` Note that an exponent of 0 gives no falloff

ILB_FO_MAX_RANGE Computes falloff as `(max((maxRange - distance), 0) / maxRange) ^ exponent`

5.6.3 Function Documentation

5.6.3.1 ILBStatus ILBAddLightLightLinks (ILBLightHandle *light*, ILBLightLinkMode *mode*, const ILBInstanceHandle ∗ *instances*, int32 *count*)

Adds a light centric light link list

Parameters:

light the light source to add light links to

mode sets whether to link or unlink the light to the instances

instances an array of instances to link to the light

count the number of instances present in the instances array

Returns:

The result of the operation.

5.6.3.2 ILBStatus ILBCreateAreaLight (ILBSceneHandle *scene*, ILBConstString *name*, const ILBMatrix4x4 ∗ *transform*, const ILBLinearRGB ∗ *intensity*, ILBLightHandle ∗ *light*)

Add an area light to the scene It points in negative Y direction by default, use the matrix to change its direction. The area light extends -1.0 to 1.0 in the X/Z dimensions, use scaling to control its area

Parameters:

scene the scene the light should be a part of

name the name of the light, must be unique within the scene.

transform the object space to world space transform for the light. It controls the area of the light as well.

intensity the colored intensity of the light source

light the handle to store the generated light source in

Returns:

The result of the operation.

5.6.3.3 ILBStatus **ILBCreateDirectionalLight** (ILBSceneHandle ***scene***, ILBConstString ***name***, **const** ILBMatrix4x4 ∗ ***transform***, **const** ILBLinearRGB ∗ ***intensity***, ILBLightHandle ∗ ***light***)

Add a directional light to the scene It points in negative Y direction by default, use the matrix to change its direction.

Parameters:

scene the scene the directional light should be a part of

name the name of the directional light, must be unique within the scene.

transform the object space to world space transform for the light.

intensity the colored intensity of the light source

light the handle to store the generated light source in

Returns:

The result of the operation.

5.6.3.4 ILBStatus **ILBCreatePointLight** (ILBSceneHandle ***scene***, ILBConstString ***name***, **const** ILBMatrix4x4 ∗ ***transform***, **const** ILBLinearRGB ∗ ***intensity***, ILBLightHandle ∗ ***light***)

Add a point light to the scene

Parameters:

scene the scene the point light should be a part of

name the name of the point light, must be unique within the scene.

transform the object space to world space transform for the light

intensity the colored intensity of the light source

light the handle to store the generated light source in

Returns:

The result of the operation.

5.6.3.5 ILBStatus **ILBCreateSpotLight** (ILBSceneHandle ***scene***, ILBConstString ***name***, **const** ILBMatrix4x4 ∗ ***transform***, **const** ILBLinearRGB ∗ ***intensity***, ILBLightHandle ∗ ***light***)

Add a spot light to the scene. It points in the negative Y-direction by default, use the matrix to point it in a different direction. The default cone angle is 90 degrees.

Parameters:

scene the scene the light should be a part of

name the name of the light, must be unique within the scene.

transform the object space to world space transform for the light.

intensity the colored intensity of the light source

light the handle to store the generated light source in

Returns:

The result of the operation.

5.6.3.6 ILBStatus **ILBCreateWindowLight** (ILBSceneHandle ***scene***, ILBConstString ***name***, **const** ILBMatrix4x4 ∗ ***transform***, **const** ILBLinearRGB ∗ ***intensity***, ILBLightHandle ∗ ***light***)

Add a window light to the scene It points in negative Y direction by default, use the matrix to change its direction. The window light extends -1.0 to 1.0 in the X/Z dimensions, use scaling to control its area

Parameters:

scene the scene the light should be a part of

name the name of the light, must be unique within the scene.

transform the object space to world space transform for the light. It controls the area of the light as well

intensity the colored intensity of the light source

light the handle to store the generated light source in

Returns:

The result of the operation.

5.6.3.7 ILBStatus **ILBSetCastShadows** (ILBLightHandle ***light***, ILBBool ***castShadows***)

Flags whether the light cast shadows or not. Disabled by default

Parameters:

light the light source in question.

castShadows sets if shadow casting should be enabled or not

Returns:

The result of the operation.

5.6.3.8 ILBStatus **ILBSetFalloff** (ILBLightHandle ***light***, ILBFalloffType ***type***, **float *exponent***, **float *cutoff***, ILBBool ***clampToOne***)

Sets the falloff for a light source. Not valid for directional lights. By default falloff is disabled. By default clamping is enabled.

Parameters:

light the light source to set falloff for

type the falloff type to use

exponent sets the exponent for the falloff

cutoff sets the influence range for the light source. It affects both falloff types, for exponent it's a hard cutoff where the light stops affecting at all. For max range it sets where the light intensity fades to zero

clampToOne sets whether to clamp the falloff to be lower or equal to one. If set to false, the falloff is allowed to scale the intensity of the light up as well as down.

Returns:

The result of the operation.

5.6.3.9 ILBStatus **ILBSetIntensityScale** (ILBLightHandle ***light***, **float *directIntensity***, **float *indirectIntensity***)

Sets scale for direct and indirect light intensity for a light source.

Parameters:

light the light source to set intensities

directIntensity direct light intensity

indirectIntensity indirect light intensity

Returns:

The result of the operation.

5.6.3.10 ILBStatus **ILBSetLightProjectedTexture** (ILBLightHandle ***light***, ILBTextureHandle ***texture***)

Sets a projected texture for a light source (gobo).

Only works for spot lights

Parameters:

light light to add gobo on

texture texture to use as gobo

Returns:

The result of the operation.

5.6.3.11 ILBStatus **ILBSetLightRampEntry** (ILBLightHandle ***light***, **float** ***position***, **const** ILBLinearRGB ∗ ***value***)

Adds a light ramp entry for falloff calculation. The ramp extends from 0 to 1 in light space, use the transformation matrix to control the scale.

Works on point and spot lights.

Parameters:

light light source to manipulate

position position in the ramp. Must be greater than 0 and greater than the last position.

value color of the given position in the ramp.

Returns:

The result of the operation.

5.6.3.12 ILBStatus **ILBSetShadowAngle** (ILBLightHandle ***light***, **float** ***angleRadians***)

Sets the angle covered of the sky for a directional light or window light for shadow casting purposes. The angle is 0 by default

Parameters:

light the light source to set the radius on.

angleRadians the angle in radians.

Returns:

The result of the operation.

5.6.3.13 ILBStatus **ILBSetShadowRadius** (ILBLightHandle ***light***, **float** ***radius***)

Sets a radius for the light source as a shadow caster. Only valid for point and spot lights. The radius is 0 by default.

Parameters:

light the light source to set the radius on.

radius the radius.

Returns:

The result of the operation.

5.6.3.14 ILBStatus **ILBSetShadowSamples** (ILBLightHandle ***light***, **int32** ***samples***)

Sets the maximum number of shadow samples for the light source. Set to 1 by default

Parameters:

light the light source to set the radius on.

samples the number of samples.

Returns:

The result of the operation.

5.6.3.15 ILBStatus **ILBSetSpotlightCone** (ILBLightHandle ***light***, **float** ***angleRadians***, **float** ***penumbraAngleRadians***, **float** ***penumbraExponent***)

Sets the cone angle for a spotlight. The cone is given in radians for the entire cone (as opposed to the angle towards the forward direction). The penumbra angle is the angle from the edge of the cone over which the intensity falls off to zero. The effective spread of the cone is max(angleRadians, angleRadians + 2∗penumbraAngleRadians) since the penumbra angle can be both positive and negative. The default cone angle is PI / 2 (90 degrees), The default is penumbra angle 0 The default penumbra exponent is 1

Parameters:

light the light source to set cone angle for

angleRadians the angle in radians for the cone

penumbraAngleRadians the angle of the penumbra of the spot light. It's given as the difference from the cone angle and can be both negative and positive.

penumbraExponent the exponent for the gradient in the penumbra

Returns:

The result of the operation.

5.7 beastmanager.h File Reference

```
#include "beastapitypes.h"
```

Defines

- #define ILB_BEAST_INTERFACE_VERSION 4

Enumerations

- enum ILBCacheScope { ILB_CS_GLOBAL, ILB_CS_LOCAL }
- enum ILBLogType { ILB_LT_ERROR, ILB_LT_INFO }
- enum ILBLogSink {
  ILB_LS_NULL, ILB_LS_STDOUT, ILB_LS_STDERR, ILB_LS_FILE,
  ILB_LS_DEBUG_OUTPUT }

Functions

- ILBStatus ILBSetStringEncodingImp (ILBStringEncoding encoding)
- static ILBStatus ILBCreateManager (ILBConstString cacheDirectory, ILBCacheScope cacheScope, ILBManagerHandle ∗beastManager)
- ILBStatus ILBDestroyManager (ILBManagerHandle beastManager)
- ILBStatus ILBClearCache (ILBManagerHandle beastManager)
- ILBStatus ILBSetBeastPath (ILBManagerHandle beastManager, ILBConstString beastPath)
- static ILBStatus ILBSetLogTarget (ILBLogType type, ILBLogSink sink, ILBConstString filename)

5.7.1 Detailed Description

The beast manager is the core object for all interaction with the Beast API

5.7.2 Define Documentation

5.7.2.1 #define ILB_BEAST_INTERFACE_VERSION 4

Revision number for released headers. Will increase with every public release with interface changes

5.7.3 Enumeration Type Documentation

5.7.3.1 enum ILBCacheScope

Sets the scope for the cache.

Enumerator:

ILB_CS_GLOBAL Makes the cache global. A different new beast manager using the same cache directory will be able to find cached resources

ILB_CS_LOCAL Makes the cache local. A different new beast manager using the same cache directory will not be able to find cached resources

5.7.3.2 enum ILBLogSink

Enum selecting where to route messages

Enumerator:

ILB_LS_NULL Discards messages

ILB_LS_STDOUT Routes messages to stdout

ILB_LS_STDERR Routes messages to stderr

ILB_LS_FILE Routes messages to a user specified file

ILB_LS_DEBUG_OUTPUT Routes messages to the debug output in visual studio when a debugger is connected

5.7.3.3 enum ILBLogType

Enum selecting a certain log target

Enumerator:

ILB_LT_ERROR Error messages

ILB_LT_INFO Information messages and render progress messages

5.7.4 Function Documentation

5.7.4.1 ILBStatus ILBClearCache (ILBManagerHandle *beastManager*)

Clears the cache. Will only work if there are no scenes present.

Parameters:

beastManager the beastManager to clear the cache for

Returns:

The result of the operation.

5.7.4.2 static ILBStatus ILBCreateManager (ILBConstString *cacheDirectory*, ILBCacheScope *cacheScope*, ILBManagerHandle ∗ *beastManager*) [inline, static]

Creates a Beast Manager

Parameters:

cacheDirectory sets the directory where the Beast Manager stores cached and temporary files.

cacheScope sets whether the cache is local to this beast manager or it can be reopened by another Beast Manager in the same directory.

beastManager a pointer to a Beast manager object that will receive the newly allocated handle

Returns:

The result of the operation.

5.7.4.3 ILBStatus ILBDestroyManager (ILBManagerHandle *beastManager*)

Destroys a Beast Manager

Will invalidate all resources and handles associated it as well

Parameters:

beastManager the Beast Manager to destroy

Returns:

The result of the operation.

5.7.4.4 ILBStatus ILBSetBeastPath (ILBManagerHandle *beastManager*, ILBConstString *beastPath*)

Sets where the Beast binaries are located. The default search order is:

1. The bin directory of where the environment variable BEAST_ROOT points. I.E. BEAST_-ROOT\bin. 2. The directory the beast dll is located in. When calling this, all other search paths are disregarded.

Parameters:

beastManager the BeastManager to set the root

beastPath the path to the Beast binaries

Returns:

ILB_ST_SUCCESS if everything went ok. ILB_ST_FILE_IO_ERROR if the specified directory doesn't contain a valid set of Beast binaries.

5.7.4.5 static ILBStatus ILBSetLogTarget (ILBLogType *type*, ILBLogSink *sink*, ILBConstString *filename*) [inline, static]

Sets where log messages should be routed. Note this function is global rather than connected since some log messages happens before a beast manager may be present or known. Note, this method is not thread safe! Don't call it while other threads are using Beast

Parameters:

type the message type to route to this target

sink where to route the messages

filename the file to write the log info to. Only used if sink is ILB_LS_FILE

5.7.4.6 ILBStatus ILBSetStringEncodingImp (ILBStringEncoding *encoding*)

Sets the character type for Beast. Should generally not be called explicitly but automatically called from ILBCreateManager or ILBSetLogTarget.

Parameters:

encoding the encoding for input and output strings.

Returns:

The result of the operation.

5.8 beastmaterial.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBMaterialChannel {
  ILB_CC_DIFFUSE = 0, **ILB_CC_SPECULAR**, **ILB_CC_EMISSIVE**, **ILB_CC_TRANSPARENCY**,
  ILB_CC_SHININESS, **ILB_CC_REFLECTION**, **ILB_CC_TOTAL_CHANNELS** }

Functions

- ILBStatus ILBCreateMaterial (ILBSceneHandle scene, ILBConstString name, ILBMaterialHandle ∗material)
- ILBStatus ILBFindMaterial (ILBSceneHandle scene, ILBConstString name, ILBMaterialHandle ∗material)
- ILBStatus ILBSetMaterialColor (ILBMaterialHandle material, ILBMaterialChannel channel, const ILBLinearRGBA ∗color)
- ILBStatus ILBSetMaterialScale (ILBMaterialHandle material, ILBMaterialChannel channel, float scale)
- ILBStatus ILBSetMaterialTexture (ILBMaterialHandle material, ILBMaterialChannel channel, ILBTextureHandle texture)
- ILBStatus ILBSetMaterialUseVertexColors (ILBMaterialHandle material, ILBMaterialChannel channel)
- ILBStatus ILBSetChannelUVLayer (ILBMaterialHandle material, ILBMaterialChannel channel, ILBConstString uvLayerName)
- ILBStatus ILBSetAlphaAsTransparency (ILBMaterialHandle material, ILBBool alphaAsTransparency)
- ILBStatus ILBSetPrimaryGICorrection (ILBMaterialHandle material, float intensity, float saturation)
- ILBStatus ILBSetSecondaryGICorrection (ILBMaterialHandle material, float intensity, float saturation)
- ILBStatus ILBSetGIScale (ILBMaterialHandle material, float diffuseBoost, float emissiveScale, float specularScale)

5.8.1 Detailed Description

The api for specifying materials in beast

5.8.2 Enumeration Type Documentation

5.8.2.1 enum ILBMaterialChannel

Defines the different channels in the Beast API shading model

5.8.3 Function Documentation

5.8.3.1 ILBStatus **ILBCreateMaterial** (ILBSceneHandle ***scene***, ILBConstString ***name***, ILBMaterialHandle $*$ ***material***)

Add a material to the scene

Parameters:

scene the scene the material should be a part of

name the name of the material, must be unique within the scene.

material a pointer to the material handle where the created material should be stored

Returns:

The result of the operation.

5.8.3.2 ILBStatus **ILBFindMaterial** (ILBSceneHandle ***scene***, ILBConstString ***name***, ILBMaterialHandle $*$ ***material***)

Checks if the material exists, if it does, it returns its handle

Parameters:

scene the scene the material is a part of

name the name of the material

material a pointer to the material handle to store the result in

Returns:

ILB_ST_SUCCESS if the material was found, ILB_ST_UNKNOWN_OBJECT if it wasn't

5.8.3.3 ILBStatus **ILBSetAlphaAsTransparency** (ILBMaterialHandle ***material***, ILBBool ***alphaAsTransparency***)

Sets if the material should use the alpha value of the diffuse channel as transparency. This is disabled by default.

Parameters:

material the material to change the setting on

alphaAsTransparency the new setting

5.8.3.4 ILBStatus **ILBSetChannelUVLayer** (ILBMaterialHandle ***material***, ILBMaterialChannel ***channel***, ILBConstString ***uvLayerName***)

Sets what UV layer should be used for the texture for a color channel.

Parameters:

material the material to change texture channel for

channel the channel to set the texture on

uvLayerName the name of the UV layer on the mesh to use. This parameter can not be checked at call time and if the UV layer is not present it will fall back on default

5.8.3.5 ILBStatus **ILBSetGIScale** (ILBMaterialHandle ***material***, **float** ***diffuseBoost***, **float** ***emissiveScale***, **float** ***specularScale***)

Sets boost and scaling factors for different aspects of the material in GI calculations. Default values are all 1.0.

Parameters:

material the material to change the setting on

diffuseBoost the value to boost the diffuse value with in GI calculations

emissiveScale the value to scale the emissive value with in GI calculations

specularScale the value to scale the specular value with in GI calculations

5.8.3.6 ILBStatus **ILBSetMaterialColor** (ILBMaterialHandle ***material***, ILBMaterialChannel ***channel***, **const** ILBLinearRGBA $*$ ***color***)

Sets the color for a channel on a material. Note that color is not supported for shininess or reflectivity.

Parameters:

material the material to apply the color on

channel the channel to set the color on

color pointer to the color to set

5.8.3.7 ILBStatus **ILBSetMaterialScale** (ILBMaterialHandle ***material***, ILBMaterialChannel ***channel***, **float** ***scale***)

Sets a scale for a channel on a material Will be multiplied with both colors and textures on the channel Can also be used to set the shininess of the material. The default value of the scale is 1.0f.

Parameters:

material the material to apply the color on *channel* the channel to set the color on

scale the scale to set

5.8.3.8 ILBStatus ILBSetMaterialTexture (ILBMaterialHandle *material*, ILBMaterialChannel *channel*, ILBTextureHandle *texture*)

Sets the texture for a channel on a material. Overrides anything set with ILBSetMaterialColor (regardless if called before or after).

Parameters:

material the material to apply the texture on

channel the channel to set the texture on.

texture the texture to apply

5.8.3.9 ILBStatus ILBSetMaterialUseVertexColors (ILBMaterialHandle *material*, ILBMaterialChannel *channel*)

Sets the material to use vertex color for a channel. This overrides anything set with ILBSetMaterialColor or ILBSetMaterialTexture (regardless if called before or after)

Parameters:

material the material to apply vertex colors on

channel the channel to set the vertex colors on support multiple vertex color sets

5.8.3.10 ILBStatus ILBSetPrimaryGICorrection (ILBMaterialHandle *material*, float *intensity*, float *saturation*)

Sets color correction on the primary GI calculations on the material. Default values are 1.0 for intensity and 1.0 for saturation.

Parameters:

material the material to change the setting on

intensity the desired primary GI intensity

saturation the desired primary GI saturation

5.8.3.11 ILBStatus ILBSetSecondaryGICorrection (ILBMaterialHandle *material*, float *intensity*, float *saturation*)

Sets color correction on the secondary GI calculations on the material. Default values are 1.0 for intensity and 1.0 for saturation.

Parameters:

material the material to change the setting on

intensity the desired secondary GI intensity

saturation the desired secondary GI saturation

5.9 beastmesh.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBBeginMesh (ILBManagerHandle beastManager, ILBConstString uniqueName, ILBMeshHandle *targetMesh)
- ILBStatus ILBEndMesh (ILBMeshHandle mesh)
- ILBStatus ILBFindMesh (ILBManagerHandle beastManager, ILBConstString uniqueName, ILBMeshHandle *target)
- ILBStatus ILBEraseCachedMesh (ILBManagerHandle beastManager, ILBConstString uniqueName)
- ILBStatus ILBAddVertexData (ILBMeshHandle mesh, const ILBVec3 *vertexData, const ILBVec3 *normalData, int32 vertexCount)
- ILBStatus ILBBeginMaterialGroup (ILBMeshHandle mesh, ILBConstString materialName)
- ILBStatus ILBEndMaterialGroup (ILBMeshHandle mesh)
- ILBStatus ILBAddTriangleData (ILBMeshHandle mesh, const int32 *indexData, int32 indexCount)
- ILBStatus ILBBeginUVLayer (ILBMeshHandle mesh, ILBConstString layerName)
- ILBStatus ILBEndUVLayer (ILBMeshHandle mesh)
- ILBStatus ILBAddUVData (ILBMeshHandle mesh, const ILBVec2 *uvData, int32 count)
- ILBStatus ILBBeginColorLayer (ILBMeshHandle mesh, ILBConstString layerName)
- ILBStatus ILBEndColorLayer (ILBMeshHandle mesh)
- ILBStatus ILBAddColorData (ILBMeshHandle mesh, const ILBLinearRGBA *colorData, int32 count)
- ILBStatus ILBBeginTangents (ILBMeshHandle mesh)
- ILBStatus ILBEndTangents (ILBMeshHandle mesh)
- ILBStatus ILBAddTangentData (ILBMeshHandle mesh, const ILBVec3 *tangentData, const ILBVec3 *bitangentData, int32 count)

5.9.1 Detailed Description

The api for specifying meshes in beast

5.9.2 Function Documentation

5.9.2.1 ILBStatus **ILBAddColorData (**ILBMeshHandle ***mesh*, const** ILBLinearRGBA * ***colorData*, int32 *count*)**

Add Color data to the active color set.

Parameters:

mesh the mesh to add color data to.

colorData a pointer to an array of color data.

count the number of colors in the array

Returns:

The result of the operation.

5.9.2.2 ILBStatus **ILBAddTangentData** (ILBMeshHandle ***mesh***, **const** ILBVec3 $*$ ***tangentData***, **const** ILBVec3 $*$ ***bitangentData***, **int32** ***count***)

Adds a batch of tangents and bitangents (binormals) to a mesh. It may be called multiple times, but the total number of added tangents/bitangents may never be more than there are vertices in the mesh.

Parameters:

mesh the mesh to add tangent data on.

tangentData an array of tangents to add

bitangentData an array of bitangents to add

count the number of tangents in the tangentData and bitangentData arrays.

5.9.2.3 ILBStatus **ILBAddTriangleData** (ILBMeshHandle ***mesh***, **const int32** $*$ ***indexData***, **int32** ***indexCount***)

Add triangles to a material group. The indices refers to the vertices added with AddVertexData. The triangles should be defined so the objects outside sees it as counter clock wise to make sure the outside is visible if rendering them single sided.

Parameters:

mesh the mesh on which to add the triangles to

indexData the indices of the triangles to add

indexCount the total index count. Must be a multiply of 3 (i.e each batch must end in a complete triangle)

Returns:

The result of the operation.

5.9.2.4 ILBStatus **ILBAddUVData** (ILBMeshHandle ***mesh***, **const** ILBVec2 $*$ ***uvData***, **int32** ***count***)

Adds a batch of UV coordinates to a mesh. It may be called multiple times, but the total number of add UV's may never be more than there are vertices in the mesh.

Parameters:

mesh the mesh to add UV data on.

uvData an array of UV coordinates to add to the UV layer

count the number of UV coordinates in the uvData array.

5.9.2.5 ILBStatus **ILBAddVertexData** (ILBMeshHandle ***mesh***, **const** ILBVec3 ∗ ***vertexData***, **const** ILBVec3 ∗ ***normalData***, **int32** ***vertexCount***)

Adds a chunk of vertex data to a mesh. This can be called multiple times to keep temporary buffer bounded.

Parameters:

mesh the mesh to add vertices on.

vertexData a pointer to an array of vertex positions

normalData a pointer to an array of vertex normals

vertexCount the number of vertices and normals specified with this call. Behavior is undefined if vertexData or normalsData contains less than vertexCount vertices/normals

Returns:

The result of the operation.

5.9.2.6 ILBStatus **ILBBeginColorLayer** (ILBMeshHandle ***mesh***, ILBConstString ***layerName***)

Creates a new color layer.

Parameters:

mesh the mesh to add the color layer to

layerName the name of the layer, must be unique.

Returns:

The result of the operation.

5.9.2.7 ILBStatus **ILBBeginMaterialGroup** (ILBMeshHandle ***mesh***, ILBConstString ***materialName***)

Begins a material group

Parameters:

mesh the mesh to add a material group to *materialName* name of the default material on this group

Returns:

The result of the operation.

5.9.2.8 ILBStatus ILBBeginMesh (ILBManagerHandle *beastManager*, ILBConstString *uniqueName*, ILBMeshHandle ∗ *targetMesh*)

Begins creation of a mesh

Parameters:

beastManager the beast manager this mesh will be associated with

uniqueName a name that must be unique withing the scene. Used to look it up in the cache.

targetMesh a pointer to a Beast mesh object that will receive the created object

Returns:

The result of the operation.

5.9.2.9 ILBStatus ILBBeginTangents (ILBMeshHandle *mesh*)

Begins adding tangents and bitangents to the mesh.

Use ILBAddTangentData to add tangent and bitanget data.

Parameters:

mesh the mesh to add the tangent layer to. Must not be finalized yet.

5.9.2.10 ILBStatus ILBBeginUVLayer (ILBMeshHandle *mesh*, ILBConstString *layerName*)

Creates a new UV layer.

Use AddUVData to add UV coordinates.

Parameters:

mesh the mesh to add the UV layer to. Must not be finalized yet.

layerName the UV layer name. Must be unique within the mesh

5.9.2.11 ILBStatus **ILBEndColorLayer** (ILBMeshHandle *mesh*)

Finalizes a color layer. The total number of added colors must be the same as the vertex count in the mesh.

Parameters:

mesh the mesh to finalize the color layer on.

Returns:

The result of the operation.

5.9.2.12 ILBStatus **ILBEndMaterialGroup** (ILBMeshHandle *mesh*)

End a material group

Parameters:

mesh the mesh to end the material group on

Returns:

The result of the operation.

5.9.2.13 ILBStatus **ILBEndMesh** (ILBMeshHandle *mesh*)

Finalizes a mesh.

After this call, it's possible to create instances from the mesh.

Will fail if any material group, uvLayer or colorLayer is unfinished

Parameters:

mesh the mesh to finalize

Returns:

The result of the operation.

5.9.2.14 ILBStatus **ILBEndTangents** (ILBMeshHandle *mesh*)

Ends the tangent layer.

Will fail if not the current number of tangents is the same as the number of vertices in the mesh.

Parameters:

mesh the mesh to finalize the tangent layer on.

5.9.2.15 ILBStatus **ILBEndUVLayer** (ILBMeshHandle ***mesh***)

Ends the UV layer currently being created.

Will fail if not the current number of UV's is the same as the number of vertices in the mesh.

Parameters:

mesh the mesh to finalize the UV layer on.

5.9.2.16 ILBStatus **ILBEraseCachedMesh** (ILBManagerHandle ***beastManager***, ILBConstString ***uniqueName***)

Erases a mesh from the cache.

Will fail if there are any resources in the beast manager referring to it (typically scenes). If this function is successful handles to the erased mesh will be invalidated and cause undefined behavior if used!

Parameters:

beastManager the beast manager to erase the mesh from.

uniqueName the name of the mesh to remove

Returns:

The result of the operation.

5.9.2.17 ILBStatus **ILBFindMesh** (ILBManagerHandle ***beastManager***, ILBConstString ***uniqueName***, ILBMeshHandle ∗ ***target***)

Finds a cached mesh.

Parameters:

beastManager the beast manager to check whether the mesh is available in

uniqueName the unique name for mesh.

target the mesh handle to store the mesh in

Returns:

The result of the operation.
ILB_ST_SUCCESS if the mesh is available ILB_ST_UNKNOWN_OBJECT if the mesh is not in the cache

5.10 beastpointcloud.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBCreatePointCloud (ILBSceneHandle scene, ILBConstString name, ILBPointCloudHandle ∗pointCloud)
- ILBStatus ILBEndPointCloud (ILBPointCloudHandle pointCloud)
- ILBStatus ILBAddPointCloudData (ILBPointCloudHandle pointCloud, const ILBVec3 ∗pointData, const ILBVec3 ∗normalData, int32 pointCount)

5.10.1 Detailed Description

The api for specifying point clouds in beast

5.10.2 Function Documentation

5.10.2.1 ILBStatus **ILBAddPointCloudData** (ILBPointCloudHandle *pointCloud*, const ILBVec3 ∗ *pointData*, const ILBVec3 ∗ *normalData*, int32 *pointCount*)

Adds a chunk of point data to a point cloud. This can be called multiple times to keep temporary buffer bounded.

Parameters:

pointCloud the point cloud to add vertices to.

pointData a pointer to an array of points

normalData a pointer to an array of normals for the points

pointCount the number of points and normals specified with this call. Behavior is undefined if vertexData or normalsData contains less than vertexCount vertices/normals

Returns:

The result of the operation.

5.10.2.2 ILBStatus **ILBCreatePointCloud** (ILBSceneHandle *scene*, ILBConstString *name*, ILBPointCloudHandle ∗ *pointCloud*)

Begins creation of a Point Cloud

Parameters:

scene the scene the point cloud should be a part of *name* the name of the material, must be unique within the scene.

pointCloud a pointer to a point cloud handle to store the result in.

Returns:

The result of the operation.

5.10.2.3 ILBStatus **ILBEndPointCloud** (ILBPointCloudHandle ***pointCloud***)

Finalizes a point cloud.

After this call it's impossible to add more points and it's possible to use the point cloud in a baking.

Parameters:

pointCloud the point cloud to finalize

Returns:

The result of the operation.

5.11 beastrenderpass.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBIlluminationMode { ILB_IM_DIRECT_ONLY = 0, ILB_IM_INDIRECT_ONLY, ILB_IM_FULL, ILB_IM_FULL_AND_INDIRECT }
- enum ILBRNMBasis { ILB_RB_HL2 = 0, ILB_RB_UE3, ILB_RB_UE3_FLIPPED, ILB_RB_CUSTOM }
- enum ILBRNMAllowNegative { ILB_AN_ALLOW = 0, ILB_AN_DISALLOW, ILB_AN_DISALLOW_CULL_HORIZON }
- enum ILBAOSelfOcclusion { ILB_SO_DISABLED = 0, ILB_SO_SET_ENVIRONMENT, ILB_SO_ENABLED }
- enum ILBLightPassType { ILB_LP_LIGHTMAP = 0, ILB_LP_SHADOWMAP, ILB_LP_FULLSHADING }

Functions

- ILBStatus ILBCreateFullShadingPass (ILBJobHandle job, ILBConstString name, ILBRenderPassHandle ∗pass)
- ILBStatus ILBCreateRNMPass (ILBJobHandle job, ILBConstString name, ILBIlluminationMode mode, int32 samples, ILBRNMBasis basis, ILBRenderPassHandle ∗pass)
- ILBStatus ILBCreateLightPass (ILBJobHandle job, ILBConstString name, ILBLightPassType type, ILBRenderPassHandle ∗pass)
- ILBStatus ILBEnableSignedDistanceField (ILBRenderPassHandle pass, int32 pixelFilterSize, float maxWorldDistance)
- ILBStatus ILBCreateLightPassEntry (ILBRenderPassHandle pass, ILBLightPassEntryHandle ∗entry)
- ILBStatus ILBAddLightToPass (ILBLightPassEntryHandle entry, ILBLightHandle light)
- ILBStatus ILBAddTargetToPass (ILBLightPassEntryHandle entry, ILBTargetEntityHandle target)
- ILBStatus ILBAddFullyBakedLight (ILBRenderPassHandle pass, ILBLightHandle light)
- ILBStatus ILBCreateNormalPass (ILBJobHandle job, ILBConstString name, ILBRenderPassHandle ∗pass)
- ILBStatus ILBCreateAmbientOcclusionPass (ILBJobHandle job, ILBConstString name, float maxDistance, float coneAngle, ILBRenderPassHandle ∗pass)
- ILBStatus ILBSetAOAdaptive (ILBRenderPassHandle pass, float accuracy, float smooth)
- ILBStatus ILBSetAONumRays (ILBRenderPassHandle pass, int32 minRay, int32 maxRay)
- ILBStatus ILBSetAOContrast (ILBRenderPassHandle pass, float contrast, float scale)
- ILBStatus ILBSetAOUniformSampling (ILBRenderPassHandle pass)
- ILBStatus ILBSetAOSelfOcclusion (ILBRenderPassHandle pass, ILBAOSelfOcclusion selfOcclusion)

- ILBStatus ILBEnableAOBentNormals (ILBRenderPassHandle pass)
- ILBStatus ILBCreateIlluminationPass (ILBJobHandle job, ILBConstString name, ILBIlluminationMode mode, ILBRenderPassHandle ∗pass)
- ILBStatus ILBCreateLuaPass (ILBJobHandle job, ILBConstString name, ILBConstString scriptFile, ILBRenderPassHandle ∗pass)
- ILBStatus ILBSetLambertianClamp (ILBRenderPassHandle pass, float val)
- ILBStatus ILBSetAllowNegative (ILBRenderPassHandle pass, ILBRNMAllowNegative allow)
- ILBStatus ILBIncludeNormalComponent (ILBRenderPassHandle pass)
- ILBStatus ILBRNMMatchNormalIntensity (ILBRenderPassHandle pass)
- ILBStatus ILBNormalizeTextures (ILBRenderPassHandle pass, bool perChannel)

5.11.1 Detailed Description

Render Pass specification

5.11.2 Enumeration Type Documentation

5.11.2.1 enum ILBAOSelfOcclusion

Self Occlusion Mode

Enumerator:

ILB_SO_DISABLED Self Occluded rays will continue beyond the originating object

ILB_SO_SET_ENVIRONMENT Self Occluded rays will be set to the environment

ILB_SO_ENABLED Objects can self occlude

5.11.2.2 enum ILBIlluminationMode

Illumination Modes

Enumerator:

ILB_IM_DIRECT_ONLY Only direct illumination (no indirect illumination)

ILB_IM_INDIRECT_ONLY Only indirect illumination (no direct illumination)

ILB_IM_FULL Both direct and indirect illumination

ILB_IM_FULL_AND_INDIRECT Stores both direct+indirect and indirect separately

5.11.2.3 enum ILBLightPassType

Light Pass Type

Enumerator:

ILB_LP_LIGHTMAP Stores the incoming light in the light map.

ILB_LP_SHADOWMAP Stores the shadow mask. The individual light mask intensity will be proportional to the light source intensity.

ILB_LP_FULLSHADING Stores the full shading in the light map.

5.11.2.4 enum ILBRNMAllowNegative

Allow Negative

Enumerator:

ILB_AN_ALLOW Allows negative RNM values

ILB_AN_DISALLOW Disallows negative RNM values

ILB_AN_DISALLOW_CULL_HORIZON Disallows negative RNM values and culls lights below the horizon of each triangle

5.11.2.5 enum ILBRNMBasis

RNM Basis

Enumerator:

ILB_RB_HL2 Half-Life 2 compatible basis

ILB_RB_UE3 Unreal Engine 3 compatible basis

ILB_RB_UE3_FLIPPED Unreal Engine 3 basis in untouched order

ILB_RB_CUSTOM Allows the user to enter the basis vectors manually

5.11.3 Function Documentation

5.11.3.1 ILBStatus ILBAddFullyBakedLight (ILBRenderPassHandle *pass*, ILBLightHandle *light*)

Add a light to be fully baked to a FullAndIndirectIllumination Pass.

Parameters:

pass the illumination pass to add the light to

light the light to add

Returns:

The result of the operation.

5.11.3.2 ILBStatus **ILBAddLightToPass** (ILBLightPassEntryHandle ***entry***, ILBLightHandle ***light***)

Add a light to a light pass

Parameters:

entry the light pass entry to add the light to

light the light to add

Returns:

The result of the operation.

5.11.3.3 ILBStatus **ILBAddTargetToPass** (ILBLightPassEntryHandle ***entry***, **ILBTargetEntityHandle** ***target***)

Add an affected target entity to a light pass

Parameters:

entry the light pass entry to add the target entity to

target the target to add

Returns:

The result of the operation.

5.11.3.4 ILBStatus **ILBCreateAmbientOcclusionPass** (ILBJobHandle ***job***, ILBConstString ***name***, **float** ***maxDistance***, **float** ***coneAngle***, ILBRenderPassHandle ∗ ***pass***)

Creates an Ambient Occlusion render pass

Parameters:

job the job to add the pass to

name the name of the pass

maxDistance the maximum distance to check for occlusion. 0 for infinite.

coneAngle the cone angle. Default is 180

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.5 ILBStatus **ILBCreateFullShadingPass** (ILBJobHandle ***job***, ILBConstString ***name***, ILBRenderPassHandle ∗ ***pass***)

Creates a Full Shading render pass

Parameters:

job the job to add the pass to

name the name of the pass

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.6 ILBStatus **ILBCreateIlluminationPass** (ILBJobHandle ***job***, ILBConstString ***name***, ILBIlluminationMode ***mode***, ILBRenderPassHandle ∗ ***pass***)

Creates an Illumination render pass

Parameters:

job the job to add the pass to

name the name of the pass

mode Selects Direct Illumination Only, Indirect Illumination only or both.

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.7 ILBStatus **ILBCreateLightPass** (ILBJobHandle ***job***, ILBConstString ***name***, ILBLightPassType ***type***, ILBRenderPassHandle ∗ ***pass***)

Creates a Light render pass

Parameters:

job the job to add the pass to

name the name of the pass

type the lighting mode

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.8 ILBStatus **ILBCreateLightPassEntry** (ILBRenderPassHandle ***pass***, ILBLightPassEntryHandle ∗ ***entry***)

Creates a Light Pass Entry

Parameters:

pass the light pass to create the entry on

entry the created entry

Returns:

The result of the operation.

5.11.3.9 ILBStatus **ILBCreateLuaPass** (ILBJobHandle ***job***, ILBConstString ***name***, ILBConstString ***scriptFile***, ILBRenderPassHandle ∗ ***pass***)

Creates a LUA pass

Parameters:

job the job to add the pass to

name the name of the pass

scriptFile the file name of the script

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.10 ILBStatus **ILBCreateNormalPass** (ILBJobHandle ***job***, ILBConstString ***name***, ILBRenderPassHandle ∗ ***pass***)

Creates a Normal render pass

Parameters:

job the job to add the pass to

name the name of the pass

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.11 ILBStatus **ILBCreateRNMPass** (ILBJobHandle ***job***, ILBConstString ***name***, ILBIlluminationMode ***mode***, int32 ***samples***, ILBRNMBasis ***basis***, ILBRenderPassHandle ∗ ***pass***)

Creates an RNM render pass

Parameters:

job the job to add the pass to

name the name of the pass

mode Selects Direct Illumination Only, Indirect Illumination only or both.

samples Number of samples for non-adaptive RNM. Set to 0 samples to turn on adaptivity (recommended).

basis The RNM basis to use

pass the handle to store the generated target in

Returns:

The result of the operation.

5.11.3.12 ILBStatus **ILBEnableAOBentNormals** (ILBRenderPassHandle ***pass***)

Calculates the "bent normal" (most visible direction). If this is used, sampling cannot be adaptive. The put will contain normals in RGB and occlusion in A

Parameters:

pass the affected pass, must be an AO pass

Returns:

The result of the operation.

5.11.3.13 ILBStatus **ILBEnableSignedDistanceField** (ILBRenderPassHandle ***pass***, int32 ***pixelFilterSize***, float ***maxWorldDistance***)

Makes a light pass use signed distance field shadow maps. Each resulting baked pixel will store the distance to the closest shadow transition.

Parameters:

pass the light pass

pixelFilterSize the maximum search range (in pixels). Default value is 20.

maxWorldDistance the maximum world distance to be stored. Default value is 1.0f.

Returns:

The result of the operation.

5.11.3.14 ILBStatus **ILBIncludeNormalComponent** (ILBRenderPassHandle ***pass***)

Enables inclusion of a normal component in the RNM pass.

Parameters:

pass the pass

Returns:

The result of the operation.

5.11.3.15 ILBStatus **ILBNormalizeTextures** (ILBRenderPassHandle ***pass***, bool ***perChannel***)

Normalizes the texture values to the 0..1 range. Stores the original range per entity which can be collected with the getNormalization∗ functions.

Parameters:

pass the pass

perChannel if enabled normalization will be done individually for each channel

Returns:

The result of the operation.

5.11.3.16 ILBStatus **ILBRNMMatchNormalIntensity** (ILBRenderPassHandle ***pass***)

Scales the RNM values to the amplitude of the normal component.

Parameters:

pass the pass

Returns:

The result of the operation.

5.11.3.17 ILBStatus **ILBSetAllowNegative** (ILBRenderPassHandle ***pass***, ILBRNMAllowNegative ***allow***)

Sets whether to allow negative coefficients on an RNM pass.

Parameters:

pass the pass

allow the allow value (default is ILB_AN_DISALLOW_CULL_HORIZON)

Returns:

The result of the operation.

5.11.3.18 ILBStatus ILBSetAOAdaptive (ILBRenderPassHandle *pass*, float *accuracy*, float *smooth*)

Enables adaptivity on an AO pass

Parameters:

pass the pass to enable adaptivity on, must be an AO pass
accuracy adaptive accuracy, default is 1
smooth smooth value, default is 1

Returns:

The result of the operation.

5.11.3.19 ILBStatus ILBSetAOContrast (ILBRenderPassHandle *pass*, float *contrast*, float *scale*)

Sets the contrast and scale of an AO pass

Parameters:

pass the affected pass, must be an AO pass
contrast the desired contrast of the AO pass. Default = 1.0f
scale scale of occlusion values. Default = 1.0f

Returns:

The result of the operation.

5.11.3.20 ILBStatus ILBSetAONumRays (ILBRenderPassHandle *pass*, int32 *minRay*, int32 *maxRay*)

Sets the number of rays to use in an AO pass

Parameters:

pass the affected pass, must be an AO pass
minRay the minimum number of rays to sample for each point, default is 64
maxRay the maximum number of rays to sample for each point, default is 300

Returns:

The result of the operation.

5.11.3.21 ILBStatus **ILBSetAOSelfOcclusion** (ILBRenderPassHandle ***pass***, ILBAOSelfOcclusion ***selfOcclusion***)

Sets how the AO pass should react to self occlusion.

Parameters:

pass the affected pass, must be an AO pass

selfOcclusion the self occlusion mode. Default is ILB_SO_ENABLED.

Returns:

The result of the operation.

5.11.3.22 ILBStatus **ILBSetAOUniformSampling** (ILBRenderPassHandle ***pass***)

Enables Uniform Sampling on an AO pass. When Uniform Sampling is enabled the sampling is not cos()-weighted.

Parameters:

pass the affected pass, must be an AO pass

Returns:

The result of the operation.

5.11.3.23 ILBStatus **ILBSetLambertianClamp** (ILBRenderPassHandle ***pass***, **float *val***)

Enable lambertian clamp on an RNM pass.

Parameters:

pass the pass

val the lambertian clamp value

Returns:

The result of the operation.

5.12 beastscene.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBBeginScene (ILBManagerHandle beastManager, ILBConstString uniqueName, ILBSceneHandle ∗target)
- ILBStatus ILBReleaseScene (ILBSceneHandle scene)
- ILBStatus ILBEndScene (ILBSceneHandle scene)

5.12.1 Detailed Description

The api for specifying scenes in beast

5.12.2 Function Documentation

5.12.2.1 ILBStatus **ILBBeginScene** (ILBManagerHandle ***beastManager***, ILBConstString ***uniqueName***, ILBSceneHandle ∗ ***target***)

Begins creation of a scene.

Parameters:

beastManager the beast manager this scene will be associated with

uniqueName a unique name for the scene.

target a pointer to a Beast scene object that will receive the created object

Returns:

The result of the operation.

5.12.2.2 ILBStatus **ILBEndScene** (ILBSceneHandle ***scene***)

Finalizes this scene Any future call to modify this scene or any of its objects will fail

Parameters:

scene the scene to finalize

Returns:

The result of the operation.

5.12.2.3 ILBStatus **ILBReleaseScene** (ILBSceneHandle *scene*)

Releases the scene data.

All handles created in this scene will be invalid after this call.

Parameters:

scene the scene to to release.

5.13 beaststring.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBGetLength (ILBStringHandle string, int32 *length)
- ILBStatus ILBCopy (ILBStringHandle string, ILBString target, int32 length)
- ILBStatus ILBReleaseString (ILBStringHandle string)

5.13.1 Detailed Description

Beast strings is objects encapsulating strings returned from Beast API functions

5.13.2 Function Documentation

5.13.2.1 ILBStatus ILBCopy (ILBStringHandle *string*, ILBString *target*, int32 *length*)

Copies the contained string to a buffer. The length is defined in the same way as in ILBGetLength

Parameters:

string the string to copy from

target a buffer to copy the string to.

length the size of the buffer. Specified as the number of characters (that may be different for different encodings) as opposed to number of bytes.

Returns:

The result of the operation.

5.13.2.2 ILBStatus ILBGetLength (ILBStringHandle *string*, int32 * *length*)

Returns the length of the string in number of characters.

The size of each character is specified by the currently used string encoding. I.E if you get 5 as length and use utf 16 your string should be 10 bytes large. Note the length includes the terminating 0. This method shall never return 0, an empty string is returned as a single termination character

Parameters:

string the string to query length from

length a pointer to an integer receiving the length

Returns:

The result of the operation.

5.13.2.3 ILBStatus ILBReleaseString (ILBStringHandle *string*)

Releases a string object.

Note that strings are not managed through the Beast Manager so it must be released manually or it will be a memory leak.

Parameters:

string the string to release.

Returns:

The result of the operation.

5.14 beasttarget.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBCreateTextureTarget (ILBJobHandle job, ILBConstString name, int32 width, int32 height, ILBTargetHandle ∗target)
- ILBStatus ILBCreateAtlasedTextureTarget (ILBJobHandle job, ILBConstString name, int32 maxWidth, int32 maxHeight, int32 maxTextures, ILBTargetHandle ∗target)
- ILBStatus ILBSetAtlasAlignment (ILBTargetHandle target, int32 alignment)
- ILBStatus ILBSetAtlasPadding (ILBTargetHandle target, int32 padding)
- ILBStatus ILBEnableAtlasSpatial (ILBTargetHandle target)
- ILBStatus ILBCreateVertexTarget (ILBJobHandle job, ILBConstString name, ILBTargetHandle ∗target)
- ILBStatus ILBCreateCameraTarget (ILBJobHandle job, ILBConstString name, ILBCameraHandle camera, int32 width, int32 height, ILBTargetHandle ∗target)
- ILBStatus ILBCreatePointCloudTarget (ILBJobHandle job, ILBConstString name, ILBTargetHandle ∗target)
- ILBStatus ILBAddBakeInstance (ILBTargetHandle target, ILBInstanceHandle bakeInstance, ILBTargetEntityHandle ∗targetEntity)
- ILBStatus ILBAddBakePointCloud (ILBTargetHandle target, ILBPointCloudHandle pointCloud, ILBTargetEntityHandle ∗targetEntity)
- ILBStatus ILBGetFramebufferCount (ILBTargetHandle target, int32 ∗count)
- ILBStatus ILBGetFramebuffer (ILBTargetHandle target, ILBRenderPassHandle pass, int32 index, ILBFramebufferHandle ∗fb)
- ILBStatus ILBGetVertexbuffer (ILBTargetHandle target, ILBRenderPassHandle pass, ILBTargetEntityHandle te, ILBFramebufferHandle ∗fb)
- ILBStatus ILBAddPassToTarget (ILBTargetHandle target, ILBRenderPassHandle pass)

5.14.1 Detailed Description

The target definitions

5.14.2 Function Documentation

5.14.2.1 ILBStatus ILBAddBakeInstance (ILBTargetHandle *target*, ILBInstanceHandle *bakeInstance*, ILBTargetEntityHandle ∗ *targetEntity*)

Adds an instance to bake to a texture or vertex bake target

Parameters:

target the target to add the instance to

bakeInstance the instance to bake

targetEntity the targetEntity for this instance. Can be 0 if you don't care

Returns:

The result of the operation.

5.14.2.2 ILBStatus **ILBAddBakePointCloud** (ILBTargetHandle ***target***, ILBPointCloudHandle ***pointCloud***, **ILBTargetEntityHandle** ∗ ***targetEntity***)

Adds a point cloud to bake

Parameters:

target the target to add the point cloud to (only works for point cloud targets)

pointCloud the point cloud to bake

targetEntity the targetEntity for this instance. Can be 0 if you don't care

Returns:

The result of the operation.

5.14.2.3 ILBStatus **ILBAddPassToTarget** (ILBTargetHandle ***target***, ILBRenderPassHandle ***pass***)

Add a Render Pass to a target

Parameters:

target the target to add the pass to

pass the pass to add to the target

Returns:

The result of the operation.

5.14.2.4 ILBStatus **ILBCreateAtlasedTextureTarget** (ILBJobHandle ***job***, ILBConstString ***name***, **int32** ***maxWidth***, **int32** ***maxHeight***, **int32** ***maxTextures***, ILBTargetHandle ∗ ***target***)

Adds an atlased texture baking target to a job

Parameters:

job the job to add the target to

name the name of the target

maxWidth the maximum width in pixels of each generated texture *maxHeight* the maximum height in pixels of each generated texture

maxTextures the maximum number of generated textures. 0 means don't care.

target the handle to store the generated target in

Returns:

The result of the operation.

5.14.2.5 ILBStatus **ILBCreateCameraTarget** (ILBJobHandle ***job***, ILBConstString ***name***, ILBCameraHandle ***camera***, int32 ***width***, int32 ***height***, ILBTargetHandle ∗ ***target***)

Adds a camera render target to a job

Parameters:

job the job to add the target to

name the name of the target

camera handle to the camera to render from

width the width in pixels of the image

height the height in pixels of the image

target the handle to store the generated target in

Returns:

The result of the operation.

5.14.2.6 ILBStatus **ILBCreatePointCloudTarget** (ILBJobHandle ***job***, ILBConstString ***name***, ILBTargetHandle ∗ ***target***)

Adds a point cloud target to a job

Parameters:

job the job to add the target to

name the name of the target

target the handle to store the generated target in

Returns:

The result of the operation.

5.14.2.7 ILBStatus **ILBCreateTextureTarget** (ILBJobHandle *job*, ILBConstString *name*, int32 *width*, int32 *height*, ILBTargetHandle ∗ *target*)

Adds a texture baking target to a job

Parameters:

job the job to add the target to

name the name of the target

width the width in pixels of the texture target

height the height in pixels of the texture target

target the handle to store the generated target in

Returns:

The result of the operation.

5.14.2.8 ILBStatus **ILBCreateVertexTarget** (ILBJobHandle *job*, ILBConstString *name*, ILBTargetHandle ∗ *target*)

Adds a vertex baking target

Parameters:

job the job to add the target to

name the name of the target

target the handle to store the generated target in

Returns:

The result of the operation.

5.14.2.9 ILBStatus **ILBEnableAtlasSpatial** (ILBTargetHandle *target*)

Enables packing spatially close objects into the same texture

Parameters:

target the target

Returns:

The result of the operation.

5.14.2.10 ILBStatus **ILBGetFramebuffer** (ILBTargetHandle ***target***, ILBRenderPassHandle ***pass***, **int32** ***index***, ILBFramebufferHandle ∗ ***fb***)

Gets a framebuffer from a target Is only valid on targets rendering images and the target is done

Parameters:

target the target to get framebuffer from

pass the pass to get vertex data for

index of the framebuffer to get

fb pointer to the handle that should receive the framebuffer

Returns:

The result of the operation.

5.14.2.11 ILBStatus **ILBGetFramebufferCount** (ILBTargetHandle ***target***, **int32** ∗ ***count***)

Gets the number of framebuffers associated with this target Is only valid on targets rendering images and if the target is done

Parameters:

target the target to get the count for

count a pointer to the variable to receive the framebuffer count

Returns:

The result of the operation.

5.14.2.12 ILBStatus **ILBGetVertexbuffer** (ILBTargetHandle ***target***, ILBRenderPassHandle ***pass***, **ILBTargetEntityHandle** ***te***, ILBFramebufferHandle ∗ ***fb***)

Gets a framebuffer with vertex data from a target Is only valid on targets rendering vertex data and the target is done

Parameters:

target the target to get vertex buffer from

pass the pass to get vertex data for

te the target entity to get vertex data for

fb pointer to the handle that should receive the framebuffer

Returns:

The result of the operation.

5.14.2.13 ILBStatus **ILBSetAtlasAlignment** (ILBTargetHandle ***target***, **int32** ***alignment***)

Sets the alignment on an atlased texture target

Parameters:

target the target

alignment the alignment

Returns:

The result of the operation.

5.14.2.14 ILBStatus **ILBSetAtlasPadding** (ILBTargetHandle ***target***, **int32** ***padding***)

Sets the padding on an atlased texture target

Parameters:

target the target

padding the padding

Returns:

The result of the operation.

5.15 beasttargetentity.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBSetBakeUVSet (ILBTargetEntityHandle target, ILBConstString uvName)
- ILBStatus ILBSetUVTransform (ILBTargetEntityHandle target, const ILBVec2 ∗offset, const ILBVec2 ∗scale)
- ILBStatus ILBSetBakeResolution (ILBTargetEntityHandle target, int32 width, int32 height)
- ILBStatus ILBSetTexelScale (ILBTargetEntityHandle target, float scale)
- ILBStatus ILBGetAtlasInformation (ILBTargetEntityHandle te, int32 ∗framebufferIndex, ILBVec2 ∗offset, ILBVec2 ∗scale)
- ILBStatus ILBGetNormalizationData (ILBTargetEntityHandle entity, ILBRenderPassHandle pass, int channel, float ∗minValue, float ∗maxValue)

5.15.1 Detailed Description

Target entities, the relationship between an instance and a bake target.

5.15.2 Function Documentation

5.15.2.1 ILBStatus ILBGetAtlasInformation (ILBTargetEntityHandle *te*, int32 ∗ *framebufferIndex*, ILBVec2 ∗ *offset*, ILBVec2 ∗ *scale*)

Gets the atlas information for a target entity Only valid on Atlas texture target entities and texture target entities

Parameters:

te the target entity to get atlas information for

framebufferIndex a pointer to write where the index of the framebuffer

offset a pointer to write the offset in uv space of the atlased object.

scale a pointer to write the scale in uv space of the atlased object.

Returns:

The result of the operation.

5.15.2.2 ILBStatus ILBGetNormalizationData (ILBTargetEntityHandle *entity*, ILBRenderPassHandle *pass*, int *channel*, float ∗ *minValue*, float ∗ *maxValue*)

Returns the minimum/maximum value in the selected entity. Used to scale the LDR values read from the frame buffer to preserve the dynamic range.

The function will fail if the pass has not enabled normalization.

Parameters:

entity the entity for which scale factors should be returned.

pass the render pass to get normalization values from

channel the framebuffer channel to use (ignored for global normalization)

minValue the minimum value for the framebuffer/entity.

maxValue the maximum value for the framebuffer/entity.

Returns:

The result of the operation.

5.15.2.3 ILBStatus ILBSetBakeResolution (ILBTargetEntityHandle *target*, int32 *width*, int32 *height*)

Sets the requested resolution for the target entity.

Only valid for atlased targets. Will fail if the resolution is higher than the maximum resolution of the atlased target.

Parameters:

target the target entity to set the resolution on

width the requested width

height the requested height

Returns:

The result of the operation

5.15.2.4 ILBStatus ILBSetBakeUVSet (ILBTargetEntityHandle *target*, ILBConstString *uvName*)

Sets the uv set to use when baking the target entity. Will use default if the uv set is not present in the instance. Only valid for texture and atlased target entities

Parameters:

target the target entity to set the uv set for *uvName* the name of the uv set to use

Returns:

The result of the operation

5.15.2.5 ILBStatus ILBSetTexelScale (ILBTargetEntityHandle *target*, float *scale*)

Sets the relationship between world space coordinates and texels for an atlased target instance. Only valid for atlased target instance. Will fail if the resolution is higher than the maximum resolution of the atlased target.

Parameters:

target the target entity to set the resolution on

scale the number of texels each world space unit should cover

Returns:

The result of the operation

5.15.2.6 ILBStatus ILBSetUVTransform (ILBTargetEntityHandle *target*, const ILBVec2 ∗ *offset*, const ILBVec2 ∗ *scale*)

Sets the a uv transform for a bake shape. It will place the shape in a specific location of the texture. Only valid for texture target entities

Parameters:

target the target entity to set the uv transform on

offset the offset in uv space for the object

scale the scale in uv space for the object

Returns:

The result of the operation

5.16 beasttexture.h File Reference

```
#include "beastapitypes.h"
```

Enumerations

- enum ILBImageGammaType { ILB_IG_GAMMA }
- enum ILBPixelFormat {

  ILB_PF_MONO_FLOAT, ILB_PF_RGB_FLOAT, ILB_PF_RGBA_FLOAT, ILB_PF_MONO_BYTE,

  ILB_PF_RGB_BYTE, ILB_PF_RGBA_BYTE }

Functions

- ILBStatus ILBReferenceTexture (ILBManagerHandle beastManager, ILBConstString uniqueName, ILBConstString filename, ILBTextureHandle *target)
- ILBStatus ILBBeginTexture (ILBManagerHandle beastManager, ILBConstString uniqueName, int32 width, int32 height, ILBPixelFormat inputFormat, ILBTextureHandle *target)
- ILBStatus ILBFindTexture (ILBManagerHandle beastManager, ILBConstString uniqueName, ILBTextureHandle *target)
- ILBStatus ILBEraseCachedTexture (ILBManagerHandle beastManager, ILBConstString uniqueName)
- ILBStatus ILBAddPixelDataHDR (ILBTextureHandle texture, const float *data, int32 pixelCount)
- ILBStatus ILBAddPixelDataLDR (ILBTextureHandle texture, const unsigned char *data, int32 pixelCount)
- ILBStatus ILBEndTexture (ILBTextureHandle texture)
- ILBStatus ILBSetInputGamma (ILBTextureHandle texture, ILBImageGammaType type, float gamma)

5.16.1 Detailed Description

The beast texture function definitions

5.16.2 Enumeration Type Documentation

5.16.2.1 enum ILBImageGammaType

Gamma for input pixels data

Enumerator:

ILB_IG_GAMMA Gamma ramp, will always be combined with a gamma value

5.16.2.2 enum ILBPixelFormat

Format for pixel data

Enumerator:

ILB_PF_MONO_FLOAT Monochrome floating point pixels

ILB_PF_RGB_FLOAT Color floating point pixels

ILB_PF_RGBA_FLOAT Color with alpha floating point pixels

ILB_PF_MONO_BYTE Monochrome byte pixels

ILB_PF_RGB_BYTE Color byte pixels

ILB_PF_RGBA_BYTE Color with alpha byte pixels

5.16.3 Function Documentation

5.16.3.1 ILBStatus ILBAddPixelDataHDR (ILBTextureHandle *texture*, const float ∗ *data*, int32 *pixelCount*)

Adds pixels to a texture. This function should only be used on textures using FLOAT pixel formats. Pixel data is treated as a linear array of pixels line by line. It may be called multiple times avoid having to replicate the entire image in Beast format, but the total number of pixels must not exceed what was specified in beginTexture.

The lines are given from the bottom and up, the lines are stored left to right.

Parameters:

texture the texture to add the pixels for

data the pixel data to add. Note this must correspond to the pixel format specified in the beginTexture call

pixelCount the number of pixel (not data values) specified in this batch.

Returns:

The result of the operation.

5.16.3.2 ILBStatus ILBAddPixelDataLDR (ILBTextureHandle *texture*, const unsigned char ∗ *data*, int32 *pixelCount*)

Adds pixels to a texture. This function should only be used on textures using BYTE pixel formats. Pixel data is treated as a linear array of pixels line by line. It may be called multiple times avoid having to replicate the entire image in Beast format, but the total number of pixels must not exceed what was specified in beginTexture.

The lines are given from the bottom and up, the lines are stored left to right.

Parameters:

texture the texture to add the pixels for *data* the pixel data to add. Note this must correspond to the pixel format specified in the beginTexture call

pixelCount the number of pixels (not data values) specified in this batch.

Returns:

The result of the operation.

5.16.3.3 ILBStatus **ILBBeginTexture** (ILBManagerHandle ***beastManager***, ILBConstString ***uniqueName***, **int32** ***width***, **int32** ***height***, ILBPixelFormat ***inputFormat***, ILBTextureHandle ∗ ***target***)

Begins creation of a texture.

Parameters:

beastManager the beast manager this texture will be associated with

uniqueName a unique name for the texture.

width the width of the texture

height the height of the texture

inputFormat the pixel format you intend to use to input data. This will not necessarily be the same format as Beast choose to save the image in.

target a pointer to a ILBTextureHandle that will receive the created texture

Returns:

The result of the operation.

5.16.3.4 ILBStatus **ILBEndTexture** (ILBTextureHandle ***texture***)

Finalizes creation of a texture. Will fail unless it has got the width ∗ height pixels added

Parameters:

texture the texture to finalize

Returns:

The result of the operation.

5.16.3.5 ILBStatus **ILBEraseCachedTexture** (ILBManagerHandle ***beastManager***, ILBConstString ***uniqueName***)

Erases a texture from the cache. Will fail if there are any resources. in the beast manager referring to it (typically scenes). If this function is successful handles to the erased texture will be invalidated and cause undefined behavior if used!

Parameters:

beastManager the beast manager to erase the mesh from.

uniqueName the name of the texture to remove

Returns:

The result of the operation.

5.16.3.6 ILBStatus ILBFindTexture (ILBManagerHandle *beastManager*, ILBConstString *uniqueName*, ILBTextureHandle ∗ *target*)

Finds a cached texture.

Parameters:

beastManager the beast manager to check whether the texture is available in

uniqueName the unique name for texture.

target the texture handle to store the mesh in

Returns:

The result of the operation. ILB_ST_SUCCESS if the texture is available ILB_ST_-UNKNOWN_OBJECT if the texture is not in the cache

5.16.3.7 ILBStatus ILBReferenceTexture (ILBManagerHandle *beastManager*, ILBConstString *uniqueName*, ILBConstString *filename*, ILBTextureHandle ∗ *target*)

References in an external texture.

Note, a referenced texture doesn't need to a call to ILBEndTexture!

Parameters:

beastManager the beast manager this texture will be associated with

uniqueName A unique name for the texture.

filename the path to the file to use

target a pointer to a ILBTextureHandle that will receive the new created texture

Returns:

The result of the operation.

5.16.3.8 ILBStatus **ILBSetInputGamma** (ILBTextureHandle ***texture***, ILBImageGammaType ***type***, **float *gamma***)

Sets what gamma encoding colors in ILBAddPixelDataLDR has.

Must executed before calling ILBAddPixelDataLDR and it's only valid on textures with an LDR pixel format. By default it's set to ILB_IG_GAMMA with gamma 2.2

Parameters:

texture the texture to set gamma on

type the gamma ramp type

gamma the gamma value.

Returns:

The result of the operation.

5.17 beastutils.h File Reference

```
#include "beastapitypes.h"
```

Functions

- ILBStatus ILBErrorToString (ILBStatus error, ILBStringHandle ∗targetString)
- ILBStatus ILBGetExtendErrorInformation (ILBStringHandle ∗targetString)
- ILBStatus ILBDumpMemoryStats ()

5.17.1 Detailed Description

Utility functions for the Beast API

5.17.2 Function Documentation

5.17.2.1 ILBStatus ILBDumpMemoryStats ()

Dumps the memory stats of the dll to the Debug console and puts the API in an undefined state. NEVER CALL ANY OTHER BEAST API FUNCTIONS AFTER THIS! Only works on debug builds, this means that it CAN only be used internally for now.

Returns:

ILB_ST_SUCCESS if there are no leaks or it's called running from a Release build

5.17.2.2 ILBStatus ILBErrorToString (ILBStatus *error*, ILBStringHandle ∗ *targetString*)

Converts an error code into a string for human readable error reporting.

Parameters:

error the error code to convert

targetString the string to receive the message.

Returns:

Result of the operation

5.17.2.3 ILBStatus **ILBGetExtendErrorInformation** (ILBStringHandle ∗ *targetString*)

Returns the last error that happened in this thread.

Error strings returned from this function may potentially be invalidated and undefined by other calls to the beast api from this thread.

Parameters:

targetString the string to receive the message.

Returns:

Result of the operation