

mental ray[®] Functional Overview

White Paper

Document version 1.5
June 13, 2007

Copyright Information

Copyright © 1986-2007 mental images GmbH, Berlin, Germany.

All rights reserved.

This document is protected under copyright law. The contents of this document may not be translated, copied or duplicated in any form, in whole or in part, without the express written permission of mental images GmbH.

The information contained in this document is subject to change without notice. mental images GmbH and its employees shall not be responsible for incidental or consequential damages resulting from the use of this material or liable for technical or editorial omissions made herein.

mental images®, mental ray®, mental matter®, mental mill™, mental queue™, mental q™, mental world™, mental map™, mental earth™, mental mesh™, mental™, Reality™, RealityServer®, RealityPlayer®, RealityDesigner®, MetaSL™, Meta™, Meta Shading™, Meta Node™, Phenomenon™, Phenomena™, Phenomenon Creator®, Phenomenon Editor™, Phenomill™, Phenograph™, neuray™, iray®, imatter®, Cybernator™, 3D Cybernator™, Shape-By-Shading®, SPM®, NRM™, and rendering imagination visible™ are trademarks or, in some countries, registered trademarks of mental images GmbH, Berlin, Germany.

Other product names mentioned in this document may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Table of Contents

Introduction	1
Applications	1
Architecture	2
Functionality	4
Rendering	4
Color and Shading	6
Geometry	9
Integration	11
Levels of Integration	11
Interactivity	11
References	12
A. Feature List	13

Introduction

mental ray® generates images of outstanding quality and unsurpassed realism. It achieves scalable performance through the exploitation of parallelism both on multiprocessor machines and across networks of machines. Its generality allows for a wide range of applications, including the creation of state of the art visual effects for movies, full length feature animations, visualization of automotive and other virtual models with accurate lighting, physically correct simulation of architectural design, as well as content creation for games.

mental ray is designed for the efficient rendering of images on multi-processor machines and networks of machines, as well as for a tight but flexible integration as a software library in 3D content creation and CAD systems.

mental ray comes with a number of standard shader packages, many of them in source code. They range from shaders for basic illumination tasks and phenomenon construction to comprehensive and optimized packages for common applications, like shaders for typical materials used in architecture and design, subsurface scattering and car paint shaders, as well as physical sun and sky lighting.

Further information about the usage, functionality, and integration of mental ray can be found in [Driemeyer 05 a, Driemeyer 05 b].

Applications

mental ray is used in a wide range of applications. This section lists a number of typical applications:

CAD Visualization

mental ray provides photorealistic design visualization of product data in CAD and styling systems.

The comprehensive support for trimmed NURBS surfaces, surface connectivity, and hierarchical subdivision surfaces offers the geometry handling capabilities required by CAD systems. Various controllable approximation settings provide any desirable geometric accuracy. Large geometric quantities can be handled efficiently, including support for heavy instancing of objects.

The ray tracing capabilities allow realistic views of the model in arbitrary surroundings, for example using car paint shaders and HDR images to visualize the realistic appearance of a car body in a given environment.

Visual Effects

mental ray is used for the creation of the most demanding visual effects in the movie industry. With its visual quality, scalability, and complete ability to customize all aspects of shading and visual elements, it offers all functionalities required for the creation of any imaginable visual effect.

Hundreds of thousands of realistic virtual characters can be included in a single final frame using mental ray's procedural, on demand, object creation features. 3D motion blur can be computed efficiently using the rasterizer rendering mode for fast, high quality first-hit rendering. This can be combined with global illumination and ray tracing simulations to achieve true virtual cinematography.

Full support for high dynamic range imaging (HDRI) and arbitrarily many named frame buffers provide the basis for seamless combination of rendered elements with live action shots.

Feature Animation

mental ray is equally well suited for the creation of full length feature animations. Any particular look can be achieved by using custom shader plugins.

Special geometric primitives and rendering modes for hair and fur, as well as advanced rendering effects like subsurface scattering for simulation of skin, allow to create and render visually compelling characters efficiently.

Game Creation

mental ray is a powerful tool to create realistic lighting and shading setups for games. Using the light mapping features of mental ray, very complex and physically correct shading and lighting calculations can be performed in mental ray, and the resulting information is output to texture maps or per-vertex color data which are then used in the game engine ("texture baking").

Architectural Design

The global illumination features in mental ray provide physically correct lighting simulations, and enable the measurement of the actual lighting intensities in a building in a given lighting environment. Visualizations of architectural designs are effectively indistinguishable from photographs due to the correct simulation of direct and indirect illumination.

Lighting Design

Light profiles, spectral rendering with more accurate color representations based on a larger number of light frequency samples, and color spaces which are visually more precise than RGB, can be used to correctly simulate the optical properties of virtual models.

Visualization

mental ray can equally well be exploited in other visualization applications. Support for volumetric effects and shading can be applied e.g. in fluid flow, seismic data, or medical visualization.

Architecture

Dataflow Architecture

The architecture of mental ray is founded on a network transparent scene and rendering database. Any data will be produced only when accessed whenever possible, resulting in scene data to be loaded or temporary data to be computed on demand only. This leads to most efficient memory usage, and together with built-in disk caching and garbage collection allows mental ray to render scenes with great complexity even on machines with a small amount of memory.

A job system is operating on top of the database. It manages the dependencies between jobs and database elements, and schedules creation, transport, and destruction of all data. Jobs can include rendered image tiles as well as geometry tessellations, but also arbitrary other data like custom user data.

Scene data may be changed between rendering of multiple frames of an animation. Incremental changes provide a way to perform updates just on the animated elements but re-using static components without

the need for re-computations. This accelerates animations as much as possible and can be used to render successive images at interactive frame rates.

Parallelism

The efficient support for simultaneous rendering on multiple processors and multiple hosts in a network is based on sophisticated, advanced parallel rendering algorithms in mental ray.

Multithreading takes full advantage of multiple processors, hyperthreaded, and multi-core processors available in a system, without requiring any additional licenses. Due to the thread-safe database and the job system distributing jobs to multiple threads, the performance of mental ray scales with the number of processors available.

Network parallelism allows the use of additional machines as rendering slaves. Rendering jobs are distributed to the machines, and scene and rendering database elements are transferred on demand, achieving scalable performance over networks of machines as well.

Satellites are available as an alternative mode of network parallel rendering in many OEM applications that contain mental ray. A number of rendering slaves (satellites) are made available to the machine running the application without requiring additional licenses, which effectively increases the rendering performance of the interactive application as well as the throughput in the batch rendering mode of the application.

Functionality

mental ray offers an optimal powerful implementation of all the features traditionally expected of photorealistic rendering software, together with unique functionality not found in any other rendering software. The following sections describe rendering features, color handling and shading, and geometry processing capabilities in mental ray.

Documentation on the usage of mental ray is contained in [Driemeyer 05 a]. mental ray's functionality is described in detail in [Driemeyer 05 b].

Rendering

Ray Tracing

The software is based on a ray tracing architecture, which allows for the flexible implementation of any imaginable phenomena and lighting effects, including reflections, refractions, global illumination, and subsurface scattering.

It uses an advanced BSP tree (more precisely a kd-tree) algorithm to speed up the ray intersection calculations. This structure is built on demand and cached. It can handle very large data sets and supports scalable multithreading.

Rasterizer

Besides ray tracing, mental ray also offers other rendering methods for situations where desired results can be produced much more efficiently.

A rasterizer is available for efficient first-hit rendering of directly visible objects and transparency. By separating visibility sampling and shading, high quality anti-aliasing can be provided while performing fewer of the expensive shading calculations (e.g. once per pixel). Motion blur can be computed with a relatively small performance impact, by shading once in the motion interval and carry this result along the motion path. This method is well suited e.g. for high quality cinematographic rendering.

A second scanline first-hit rendering method is also available, which is sometimes faster on smaller scenes with relatively simple shading.

Global Illumination

Global illumination is the simulation of all light inter-reflection effects in a scene. This includes indirect illumination caused by the scattering of light, and effects such as caustics and color bleeding: if a red table is next to a white wall, the white wall gets a reddish tint under typical lighting conditions. Without this reddish tint the image would look fake, even though it might be hard to point out precisely why. Global illumination effects are subtle but essential to true photorealism.

Simulation of global illumination has at least two distinct uses:

- Physically accurate simulation of the illumination in an environment, for example the light distribution inside an office building.
- Visually pleasing lighting effects for applications in the entertainment industry. Here physical accuracy is not the most important aspect; the images just have to look more believable.

mental ray offers two fundamental approaches to compute global illumination which can be used together. They differ in the way lighting information is traced through the scene, either starting from the light (photon mapping) or starting from the eye (final gathering).

The *photon mapping* technique emits small energy particles (photons) from light sources in a pre-process, traces their trajectories through the scene including reflections, refractions, and interactions with participating media, and stores the final energy distribution in a specialized 3d data structure called photon map. In the final rendering phase this global illumination contribution is added by collecting the color intensities of nearby photons.

Photon mapping supports caustics (e.g. sun light patterns on the ground of a swimming pool), participating media (e.g. shafts of light in a dusty room), color bleeding, arbitrarily many light bounces, as well as advanced material properties like glossy reflections.

Photon maps can be pre-computed and saved on disk for later reuse in subsequent renderings. Photon merging allows to reduce memory consumption for photon storage in scenes where a high number of photons must be shot to provide sufficient illumination.

The *final gathering* technique computes global illumination by tracing rays as usual from a shading point into the scene to catch light. In contrast to collecting the contribution from a distinct number of light sources it sends out many more rays into all possible directions of the hemisphere to capture illumination from other objects that interact with light (indirect lighting) like diffuse reflections (diffuse materials spread incoming light into many directions). This *exact final gathering* mode is physically accurate but also expensive to compute.

mental ray provides an efficient technique to reduce the render time dramatically while retaining the final image quality as much as possible. The full final gather tracing is performed only on distinct and well selected surface points. All other surface points quickly interpolate the global illumination contribution from nearby final gather points. The final gather points are selected according to the local surface and illumination variation and their impact on final image quality. Several final gathering modes are provided which are optimized for ease of use, fly-through animations through static environments, and full user control, respectively.

The final gather data can also be saved on disk, and refined and re-used for subsequent renderings. Final gathering gives particularly good results for diffuse surfaces with a single light scattering bounce only.

Further variations of the base algorithms provide simplified and therefore more efficient applications for global illumination effects like contact shadows (*ambient occlusion*), and importance-driven photon mapping (*importons*).

Hardware Rendering

Now that modern GPUs can execute complex programs to compute the color of each shaded pixel, mental ray can utilize this power for accelerating its high quality rendering. Many elements can be rendered entirely in hardware, and are automatically combined with elements which still need to be rendered in software.

mental ray currently exploits the advanced Cg shading language by NVIDIA [Cg] and uses OpenGL to exchange data with the graphics card. Cg versions of a large number of shaders including OEM shader libraries are available. mental mill™ [mental mill] supports the mapping of MetaSL™ shaders [MetaSL] to both C++ and Cg, which removes the need to provide and maintain separate hardware versions of shaders.

The unique implementation of hardware rendering in mental ray is the only one to provide full support for all of the following state of the art features:

- Phenomena™
- high quality anti-aliasing
- order independent transparency
- motion blur
- soft shadows
- tiling for large image resolutions and high quality anti-aliasing

A description of the hardware support in mental ray is also available in [LeFrancois 05]. For more information on mental ray Phenomena please refer to the section on Phenomena below.

Light Mapping

Light mapping, also sometimes called *texture baking*, is a method for sampling an object before rendering, and storing the results for later use. The most common application samples illumination for each point of a texture image wrapped around the object, and stores that illumination in the texture. Illumination is effectively frozen into the texture, which can later be mapped onto an object in the conventional way.

The advantage is that rendering can obtain illumination quickly from the frozen illumination, instead of computing it at rendering time. This is especially valuable for indirect illumination, which takes more time to compute than direct illumination. Such textures, called light maps, are commonly used in games and interactive visualization applications.

Light mapping is very flexible in mental ray and can be customized in light map shaders. Typical light map rendering writes output to standard textures files, which are perfectly applicable in mental ray and other capable software but especially useful with hardware rendering.

Multipass Rendering

The unique multipass rendering feature allows users to output and composite shading data in sub-pixel resolution with an arbitrary number of samples per pixel. The anti-aliasing and resolution limitations of regular compositing can be overcome this way. The compositing and merging of passes can be controlled by shaders.

Color and Shading

Shader Interface

Most aspects of shading are performed in custom plug-in functions called 'shaders'. Operations provided by shaders include:

- surface shading
- texture mapping

- procedural textures
- light sources
- customized shadows
- volumetric effects
- simulation of camera lenses
- accumulation of shading results and other data into frame buffers
- displacement mapping
- procedural geometry and other scene elements

The interface available to shaders is extremely flexible, it provides all the essential shading and illumination functions but also allows advanced shaders to access the complete rendering database as well as application data.

Phenomena

A Phenomenon™ is an encapsulated set of cooperating shaders and shader graphs with an optional set of requirements. On the outside, a Phenomenon looks just like a regular shader, with a name and external parameters called interface parameters. But on the inside it packages any number of internal shaders and rendering options that all work together to create a new shading function or even a complete visual effect.

The encapsulated shader graphs facilitate the integration of custom shader networks into applications. Complex shader graphs can be presented to the user as a single entity with a small, well defined interface and set of parameters. See also [mental mill, MetaSL].

Shadows

mental ray provides highly optimized and accurate ray traced shadows. This supports accurate sampling of soft shadows by area light sources, colored shadows, and volumetric attenuation of shadows.

Shadow maps are also available for fast generation of shadows for cases where shadow mapping techniques outperform ray traced shadows and resulting shadow quality is acceptable. Soft shadowing algorithms are supported with shadow maps. Shadow artifacts caused by self intersections are actively avoided.

The proprietary *detail shadow map* method collects more information about shadow blocking elements of the scene, by storing multiple samples per pixel and supporting colored, transparent shadows. This technique combines advantages of ray traced shadows and shadow maps and is especially useful for shadowing of fur and hair, where regular shadow maps do not provide the necessary resolution in terms of z depth and shadow map pixels.

Textures

mental ray's texture handling supports tiled texture storage and caching for very large resolution texture maps, arbitrary texture formats including high dynamic range images, high quality texture filtering, bump mapping, and high frequency displacement mapping. Arbitrary texture mapping can be implemented in shaders.

Volumes

Volumes and atmospheres (i.e. a volume surrounding all objects) are implemented in volume shaders. Volumes can interact with all aspects of shading, including shadows, global illumination, and volume caustics, caused by the scattering of light in a participating medium after being reflected or refracted. Volumes can be nested.

Frame Buffers and HDRI

Colors are internally stored in frame buffers which can be represented in a number of available formats, including various high dynamic range (HDR) formats. Arbitrarily many frame buffers can be generated in order to output multiple passes at once for later compositing or storage of custom per-pixel information.

Frame buffers can be referenced by name in the scene and in shaders. Their content is saved out to disk during rendering to be able to use arbitrarily many and arbitrarily large frame buffers without running out of memory. Efficient access is achieved by utilizing memory mapping techniques.

Textures and output images can be represented in more than 30 common image formats, including HDR formats like OpenEXR.

MetaSL and mental mill

Shaders in the shading language *MetaSL*TM created with the *mental mill*TM technology are supported by mental ray [MetaSL, mental mill]. Shaders written in MetaSL can be rendered seamlessly in software and in hardware, and can be adapted to the particular purpose independently of their origin.

mental mill provides a way to create Phenomena graphically. This solution makes it practical to use complex shader graphs containing very simple nodes called Metanodes, since the networks are combined into larger MetaSL shaders at creation time which avoids run time overheads of evaluating a large number of connected shaders.

Light Profiles

mental ray supports IES and Eulumdat light profiles. Light profiles are made available by vendors of physical lamps to describe precisely how much light is emitted in which direction. This captures the physical properties of complex light sources and allows accurate rendering of resulting illumination in synthetic scenes. Eulumdat is a European format while IES is an international standard.

Color Spaces and Spectral Rendering

RGB colors do not provide sufficient color frequency resolution for all applications, like lighting simulations or wavelength dependent refractions. Color calculations can therefore be performed in mental ray in a number of other, more accurate tristimulus color spaces, like CIE XYZ.

If this is not sufficient, color can also be represented by spectra, represented by a larger number of separate frequency samples, allowing wavelength dependent optics with high accuracy.

BRDFs

Actual measured surface characteristics can be represented in bidirectional reflectance distribution functions (BRDFs). Libraries of surfaces can be provided in this representation, and BRDFs can also

be used to manipulate surface characteristics, create artificial materials, or even to represent precomputed global illumination characteristics. mental ray provides data formats, conversions, and operations on BRDFs which can be used by BRDF shaders.

Contour Rendering

Contour lines can be an important visual cue to distinguish objects and accentuate their forms and spatial relationship. Contour lines are especially useful for cartoon style animation productions.

Contours can be placed at discontinuities of depth or surface orientation, between different materials, or where the color contrast is high. Custom contour detection can be implemented in shaders. The contour lines are anti-aliased. There can be several levels of contours created by ray tracing reflections or seen through semitransparent materials.

Geometry

Free Form Surfaces

mental ray supports free-form curves in various representations, like non-uniform rational B-spline (NURBS), Bézier, Taylor (monomial), cardinal or general basis matrix form. Any of these forms may be rational and may be of degree up to 21. Surfaces may be trimmed.

Polygons

Support for polygonal geometry includes arbitrary concave polygons and polygons with holes. Additional data can be attached on a per-vertex basis, like texture coordinates, derivative vectors, or pre-computed colors. An efficient representation of triangle-only meshes is supported which accelerates common data exchange with many modeling applications.

Hierarchical Subdivision Surfaces

Hierarchical subdivision surfaces are available in mental ray in conjunction with the *mental matter*[®] technology [mental matter]. It implements the Loop subdivision scheme for triangle meshes, and the Catmull-Clark subdivision scheme for quadrilateral meshes. Meshes of polygons with arbitrarily many polygons can also be rendered using a Catmull-Clark subdivision. Mixed triangle-quadrilateral meshes are natively supported with a special subdivision scheme not requiring any conversions.

Further features include a C++ API for the construction and manipulation of surfaces in geometry shaders, crease edges with fractional sharpness, trimming, adaptive local refinement, multiresolution editing with wavelet decomposition and variable data-dependent smoothing, and conversion of subdivision surfaces to NURBS and of NURBS to subdivision surfaces.

Approximations

Surfaces and polygons are internally tessellated to triangles. This tessellation can be adapted to the respective requirements using a number of approximation techniques, including parametric tessellations, control by edge length, distance, and curvature (angle) criteria, view dependent tessellations to pixel level or below, and so called fine approximations for memory-efficient handling of large surfaces by tessellating them in smaller portions. Fine approximations work particularly well with detailed tessellations of high frequency displacement maps.

Instancing

Geometric objects, cameras, lights, and instance groups can be instanced multiple times in a scene. Heavy instancing provides for rendering of many copies of an object at different transformations without replicating the object data in memory.

Displacement Mapping

Surfaces and polygons can be displaced along the normal using displacement maps controlled by shaders. In the presence of high frequency displacement maps mental ray takes care of tessellating the surfaces finely enough and with efficient memory usage.

Hair and Fur

Special hair primitives are provided to render hair and fur. The primitives enable the creation and rendering of large amounts of hair efficiently from spline curves. Specialized ray tracing acceleration data structures are employed in this case. Large amounts of hair can also be rendered efficiently using the rasterizer and detail shadow maps, keeping render times and memory consumption low while providing good anti-aliasing of the hairs as well as optional efficient motion blur.

Geometry Shaders, Placeholders, Assemblies

mental ray provides several mechanisms to load or create geometry on demand only. This allows to perform and optimize rendering of large scenes on machines where memory is insufficient to load the whole scene at once.

Geometry shaders are C or C++ plugins that procedurally create geometric objects and other scene elements. Objects can be created on demand with geometry shaders when a given bounding box is hit. *Placeholder* objects allow for the loading of individual objects on demand from disk just in time when a specified bounding box is hit. *Assembly* scenes can contain a complete sub scene hierarchy to be loaded on demand, which matches and supports the typical workflow for large-scale scene modeling.

These techniques enable to delete large objects from memory during rendering if necessary and reload on demand from .mi files or shader functions when needed.

Integration

Levels of Integration

mental ray can be integrated into digital content creation and design and engineering tools in three different ways, offering different levels of tightness and interactivity:

1. mental ray is available in a C++ library version for direct integration into the application. A comprehensive API allows for the control of all rendering aspects, and to pass all elements directly to mental ray. This is the tightest integration possible, and allows full scene database sharing across the application, the mental ray library, and shader plugins. The integration API is described in detail in [Driemeyer 05 b].
2. A translator application can be provided which reads the data of the content creation system and passes it to the database of the mental ray library.
This is a library integration like 1.), but does not necessarily provide optimal interactive rendering in an application. It allows for example the batch mode rendering of 3D content which is present in the proprietary format of the content creation tool.
3. A translator plugin exports the scene in mental images' .mi scene file format, which is then rendered using the standalone version of mental ray.
This is very useful for example for the operation of a render farm. It also enables the transfer or further processing of the scene data for rendering. This method is sometimes more efficient than 1.) and 2.) since no application data needs to be loaded into memory, and hence more memory may be available for rendering.
The .mi export mode is a byproduct of the integrations in 1.) and 2.) and can also be implemented just by writing an exporter from the application.

Interactivity

Integrations of type 1.) in particular result in rendering system characteristic of a very high interactivity. Advantages of a tight integration include:

- The scene can be rendered directly in memory, without going through input and output to the file system, which greatly speeds up scene preparation.
- Interactive rendering using incremental changes.
- Scene elements can be changed interactively between frames or even during rendering of a frame.
- Shaders have full access to the application data, and the application has direct access to rendering data. Shaders can for example query the application data to affect rendering. Images and other rendering output can be fed back to the application directly in memory.
- The application and mental ray can cooperate in the efficient use of resources. If for example memory is tight in the application, it can ask mental ray to release as much memory as possible, in order to keep the application operable. If on the other hand memory is tight for rendering, mental ray can prompt the application to release memory as needed or to control how mental ray should proceed if sufficient memory is not available.
- Diagnostics can be customized to the application, filtering and modifying what is presented to the user. For example, diagnostics output can be internationalized.

- With the availability of custom shaders, mental ray can mimic precisely the shading models and parameters of the application, and can even extend it. A scene set up in the application can thus be rendered without additional user intervention, but with the added rendering capabilities of mental ray.
- Rendering within an integrated mental ray library enables rendering with the licensing and protection schemes of the application, and avoids any data security issues implied by export of the scene to the file system.
- To speed up the (interactive) rendering of the application, any number of mental ray co-processes on a network of machines can be utilized from within the application. In this setup, the integrated library within the application can be relieved of performing rendering computations, and can help reducing memory consumption on the master host.

References

- [Driemeyer 05 a] Thomas Driemeyer, *Rendering with mental ray*. 3rd Edition. Springer, 2005.
- [Driemeyer 05 b] Thomas Driemeyer, Rolf Herken, *Programming mental ray*. 3rd Edition. Springer, 2005.
- [LeFrancois 05] Martin-Karl LeFrancois, “mental ray Phenomena on the GPU”, *GPU Gems II*. Chapter 13, Addison-Wesley 2005.
- [mental matter] *mental matter Modeling Library, version 3.4: Functional Overview*. Document Version 3.5. mental images GmbH, Berlin, Germany, 2004.
- [mental mill] *Functional overview of mental mill™*. mental images GmbH, Berlin, Germany, 2006.
- [MetaSL] *Design specification for MetaSL™*. mental images GmbH, Berlin, Germany, 2007.
- [Cg] Randima Fernando, Mark J. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003.

A. Feature List

Rendering

- ray tracing architecture for automatic, accurate, and general simulation of reflections, refractions, shadows, and complex illumination
- physically correct simulation of global illumination
 - photon mapping, simulating all possible light paths
 - computation of caustics using dedicated photon maps
 - optional merging of photons to reduce memory consumption
 - importance driven photon map optimizations
 - final gathering to collect indirect lighting from diffuse light bounces optionally combined with photon mapping, with several modes optimized for performance and ease of use in common cases,
 - ambient occlusion for fast simulation of contact shadows
 - participating media: volume caustics and multiple volume scattering
- anti-aliasing
 - strictly deterministic low-discrepancy sampling
 - automatic anti-aliasing of all features such as transparency, refractions, reflections, shadows, area lights, textures, caustics and volume caustics, and global illumination and multiple volume scattering
 - adaptive recursive oversampling and under-sampling (fewer than one sample per pixel taken initially, automatically refined where necessary)
- hardware rendering, supporting:
 - Phenomena
 - Cg shaders
 - order independent transparency
 - motion blur
 - antialiasing
 - tiling for large image resolutions and high quality antialiasing
 - combination of software and hardware elements
- motion blur, depth of field
 - object transformation motion blur
 - per-vertex motion deformations
 - multi-segment vertex motion blur for curved motion blur
 - full motion blur of reflections, refractions, light sources, shadows, caustics, and global illumination
 - depth of field implemented in lens shader
 - motion and depth output for post processing motion blur and depth of field
- light mapping
- multipass rendering in sub pixel sample resolution

Performance

- full dataflow architecture with caching and on demand data creation
- multithreading parallelism
- network parallel rendering on heterogeneous networks of machines
- incremental changes
- rasterizer for fast, high quality first generation rendering
- scanline rendering method
- BSP tree (kd-tree) acceleration method for fast ray tracing, with memory caching for very complex scenes
- new self-tuning and memory efficient BSP2 tree acceleration, optimized for fast ray tracing of scenes with dynamic scene manipulations by assemblies
- full use of 64-bit architectures

Color and Shading

- programmable C and C++ shaders, with full access to scene data
- shade trees and shader graphs
- encapsulated shader graphs (Phenomena)
- MetaSL shaders, both in software and hardware rendering modes
- shader types: material, texture, light, volume, shadow, atmosphere, lens, environment, photon material, photon emitter, displacement, output, contour, geometry, inheritance, lightmap, and state shaders
- volume shaders implement non-geometric volume effects
- built-in shaders
- arbitrary many and arbitrary large frame buffers, stored on disk during rendering
- arbitrary frame buffer and image data types, including color, depth, motion, normals, labels, and coverage
- HDRI (high dynamic range imaging) using RGBE, float, and half float data types
- over 30 image types for textures and output, including OpenEXR
- IES and Eulumdat light profiles
- color spaces other than RGB
- BRDFs
- spectral rendering
- ray marching for volume shading

- incremental image preview during rendering
- contour rendering
- subsurface scattering
- shaders:
 - architectural and design shaders
 - extensive base shader library
 - subsurface scattering
 - glossy reflections & refractions
 - metallic car paint
 - physically correct shaders
 - spectral rendering shaders
 - contour lines
 - depth of field
 - 2D motion blur production shaders

Shadows

- ray traced shadows
- shadow maps
- detail shadow maps
- area lights: spheres, disks, rectangles, cylinders, arbitrarily shaped geometric light objects
- shadow shaders
- volume cast shadows and volume self-shadowing

Textures

- texture tiling and caching
- over 30 texture image formats
- mipmapping and pyramid textures
- environment maps
- memory-mapped textures and memory-mapped pyramid textures
- high quality texture filtering

Geometry

- free-form surface bases: Bézier, B-spline, NURBS, Cardinal, and arbitrary basis matrix, Taylor; rational and non-rational, up to degree 21
- any number of trimming curves, hole curves, and special curves and points
- connectivity between surfaces
- polygons
- triangle lists
- hierarchical subdivision surfaces:
 - quads (Catmull-Clark scheme)
 - triangles (Loop scheme)
 - Catmull-Clark subdivision of meshes of polygons with arbitrarily many vertices
 - sharp features: creases and points
 - wavelet-based multiresolution editing and data compression/LOD (mental matter)
 - conversion from and to NURBS (mental matter)
- trimming
- approximation, controllable by parametric subdivisions, edge length, analytic distance, angle
- displacement mapping
- texture surfaces, multiple texture coordinates
- arbitrary per-primitive data in polygonal objects
- special hair primitives
- instancing
- object and instance flags, per object control of properties like visibility, shadows, reflections, etc.
- selection of lights for illumination and shadowing per instance
- placeholders
- assemblies
- on-demand loading of scene content from .mi files
- on-demand procedural creation of scene content using geometry shaders
- optional on-demand re-loading of large data from .mi file or callback

Integration

- optional library form for integration into OEM products
- translators and plugins are available for a variety of front-end applications
- C/C++ language API for entire scene description language and other features
- simple, efficient, and full-featured hierarchical scene description language
- material and parameter inheritance
- OEM specific software protection and licensing technology