Intel® Embree

High Performance Ray Tracing Kernels

Version 4.4.0
March 27, 2025

Contents

Chapter 1

Intel® Embree Overview

Intel® Embree is a high-performance ray tracing library developed at Intel, which is released as open source under the Apache 2.0 license. Intel® Embree supports x86 CPUs under Linux, macOS, and Windows; ARM CPUs on Linux and macOS; as well as Intel® GPUs under Linux and Windows.

Intel® Embree targets graphics application developers to improve the performance of photo-realistic rendering applications. Embree is optimized towards production rendering, by putting focus on incoherent ray performance, high quality acceleration structure construction, a rich feature set, accurate primitive intersection, and low memory consumption.

Embree's feature set includes various primitive types such as triangles (as well quad and grids for lower memory consumption); Catmull-Clark subdivision surfaces; various types of curve primitives, such as flat curves (for distant views), round curves (for closeup views), and normal oriented curves, all supported with different basis functions (linear, Bézier, B-spline, Hermite, and Catmull Rom); point-like primitives, such as ray oriented discs, normal oriented discs, and spheres; user defined geometries with a procedural intersection function; multi-level instancing; filter callbacks invoked for any hit encountered; motion blur including multi-segment motion blur, deformation blur, and quaternion motion blur; and ray masking.

Intel® Embree contains ray tracing kernels optimized for the latest x86 processors with support for SSE, AVX, AVX2, and AVX-512 instructions, and uses runtime code selection to choose between these kernels. Intel® Embree contains algorithms optimized for incoherent workloads (e.g. Monte Carlo ray tracing algorithms) and coherent workloads (e.g. primary visibility and hard shadow rays) as well as supports for dynamic scenes by implementing high-performance two-level spatial index structure construction algorithms.

Intel® Embree supports applications written with the Intel® Implicit SPMD Program Compiler (Intel® ISPC, `https://ispc.github.io/`) by providing an ISPC interface to the core ray tracing algorithms. This makes it possible to write a renderer that automatically vectorizes and leverages SSE, AVX, AVX2, and AVX-512 instructions.

Intel® Embree supports Intel GPUs through the SYCL open standard programming language. SYCL allows to write C++ code that can be run on various devices, such as CPUs and GPUs. Using Intel® Embree application developers can write a single source renderer that executes efficiently on CPUs and GPUs. Maintaining just one code base this way can significantly improve productivity and eliminate inconsistencies between a CPU and GPU version of the renderer. Embree supports GPUs based on the Xe HPG and Xe HPC microarchitecture, which support hardware accelerated ray tracing do deliver excellent levels of ray tracing performance.

1.1 Supported Platforms

Embree supports Windows (32-bit and 64-bit), Linux (64-bit), and macOS (64-bit). Under Windows, Linux and macOS x86 based CPUs are supported, while ARM CPUs are currently only supported under Linux and macOS (e.g. Apple M1). ARM support for Windows experimental.

Embree supports Intel GPUs based on the Xe HPG microarchitecture (Intel® Arc™ GPU) under Linux and Windows and Xe HPC microarchitecture (Intel® Data Center GPU Flex Series and Intel® Data Center GPU Max Series) under Linux.

The code compiles with the Intel® Compiler, Intel® oneAPI DPC++ Compiler, GCC, Clang, and the Microsoft Compiler. To use Embree on the GPU the Intel® oneAPI DPC++ Compiler must be used. Please see section Compiling Embree for details on tested compiler versions.

Embree requires at least an x86 CPU with support for SSE2 or an Apple M1 CPU.

1.2 Embree Support and Contact

If you encounter bugs please report them via Embree's GitHub Issue Tracker.

For questions and feature requests please write us at `embree_support@intel.com`.

To receive notifications of updates and new features of Embree please subscribe to the Embree mailing list.

1.3 Version History

1.3.1 Embree 4.4

- Added support for passing geometry data to Embree using explicit host and SYCL device memory (see `rtcSetSharedGeometryBufferHostDevice`, `rtcNewBufferHostDevice`, and other API calls with `HostDevice` suffix).
- Embree does not use SYCL shared memory anymore internally on systems without host unified memory (i.e., discrete GPUs). Therefore, memory transfers are triggered by specific Embree API calls (e.g. `rtcCommitScene`, `rtcCommitBuffer`).
- Objects of type `RTCScene` are not accessible on a SYCL device anymore. Ray queries on SYCL devices must be performed using `RTCTraversable` objects and the `rtcTraversableIntersect` and `rtcTraversableOccluded` API calls.
- Embree does not query the availability of RDRAND for its ISA detection anymore, which caused issues on some older AMD CPUs.
- Performance improvements on GPU for the two level instancing case (RTC_MAX_INSTANCE_LEVEL_COUNT 2).

1.3.2 Embree 4.3.3

- Added RTCError RTC_ERROR_LEVEL_ZERO_RAYTRACING_SUPPORT_MISSING which can indicate a GPU driver that is too old or not installed properly.
- Added the API function rtcGetDeviceLastErrorMessage to query additional information about the last RTCError returned by rtcGetDeviceError. This can be used in case device creation failed and a rtcErrorFunction could not be set up for this purpose.

- Added the API function rtcGetErrorString which returns a string representation of a given RTCError error code. This is purely meant for convenient error information reporting on the user application side.
- Performance improvements on GPU for the one level instancing case (RTC_MAX_INSTANCE_LEVEL_COUNT 1).
- Reduced the number of unnecessary GPU-CPU USM back-migrations which can increase build performance for scene with many instances on GPU.
- Started adding public CI tests for streamlining integration of external pull requests.
- Work-around for problem with unsigned Windows binaries.

1.3.3 Embree 4.3.2

- Embree now uses level zero raytracing extension to build BVH which enables forward-compatibility. On Linux, the package intel-level-zero-gpu-raytracing has to be installed in addition to the other packages listed here https://dgpu-docs.intel.com/.
- MacOS universal binary compilation now works.
- Some bugfixes for AVX512 on MacOS x86 machines.
- Known issue: It is recommended to run Embree on Intel® Data Center GPU Max Series (e.g. Intel® Data Center GPU Max 1550) with the following environment settings: NEOReadDebugKeys=1 UseKmdMigration=0
- Known issue: ISPC version of tutorials will not successfully build with MacOS universal binary compilation.

1.3.4 Embree 4.3.1

- Add missing EMBREE_GEOMETRY types to embree-config.cmake
- User defined thread count now takes precedence for internal task scheduler
- Fixed static linking issue with ze_wrapper library
- Better error reporting for SYCL platform and driver problems in embree_info and tutorial apps.
- Patch to glfw source is not applied by default anymore.
- Known issue: Running Embree on Intel® Data Center GPU Max Series with 2 tiles (e.g. Intel® Data Center GPU Max 1550) requires setting the environment variable ZE_FLAT_DEVICE_HIERARCHY=COMPOSITE.
- Known issue: Embree build using Apple Clang 15 and ARM support (via the SEE2NEON library) may cause “EXEC_BAD_INSTRUCTION” runtime exceptions. Please use Apple Clang <= 14 on macOS.

1.3.5 Embree 4.3.0

- Added instance array primitive for reducing memory requirements in scenes with large amounts of similar instances.
- Properly checks driver if L0 RTAS extension can get loaded.
- Added varying version of rtcGetGeometryTransform for ISPC.
- Fixed signature of RTCMemoryMonitorFunction for ISPC.
- Add support for ARM64 Windows platform in CMake.

1.3.6 Embree 4.2.0

- SYCL version of Embree with GPU support is no longer in beta phase.
- Improved BVH build performance on many core machines for applications that oversubscribe threads.
- Added rtcGetGeometryTransformFromScene API function that can get used inside SYCL kernels.

- No longer linking to ze_loader in SYCL mode to avoid Intel(R) oneAPI Level Zero dependency for CPU rendering.
- Releasing test package to test Embree.

1.3.7 Embree 4.1.0

- Added support for Intel® Data Center GPU Max Series.
- Added ARM64 Linux support.
- Added EMBREE_BACKFACE_CULLING_SPHERES cmake option. The new cmake option defaults to OFF.

1.3.8 Embree 4.0.1

- Improved performance for Tiger Lake, Comet Lake, Cannon Lake, Kaby Lake, and Skylake client CPUs by using 256 bit SIMD instructions by default.
- Fixed broken motion blur of RTC_GEOMETRY_TYPE_ROUND_LINEAR_CURVE geometry type.
- Fixed bvh build retry issue for TBB 2020.3
- Added support for Intel® Data Center GPU Flex Series
- Fixed issue on systems without a SYCL platform.

1.3.9 Embree 4.0.0

- This Embree release adds support for Intel® Arc™ GPUs through SYCL.
- The SYCL support of Embree is in beta phase. Current functionality, quality, and GPU performance may not reflect that of the final product. Please read the documentation section “Embree SYCL Known Issues” for known limitations.
- Embree CPU support in this release as at Gold level, incorporating the same quality and performance as previous releases.
- A small number of API changes were required to get optimal experience and performance on the CPU and GPU. See documentation section “Upgrading from Embree 3 to Embree 4” for details.
- rtcIntersect and rtcOccluded function arguments changed slightly.
- RTCIntersectContext is renamed to RTCRayQuery context and most members moved to new RTCIntersectArguments and RTCOccludedArguments structures.
- rtcFilterIntersection and rtcFilterOcclusion API calls got replaced by rtcInvokeIntersectFilterFromGeometry and rtcInvokeOccludedFilterFromGeometry API calls.
- rtcSetGeometryEnableFilterFunctionFromArguments enables argument filter functions for some geometry.
- RTC_RAY_QUERY_FLAG_INVOKE_ARGUMENT_FILTER ray query flag enables argument filter functions for each geometry.
- User geometry callbacks have to return if a valid hit was found.
- Ray masking is enabled by default now as required by most users.
- The default ray mask for geometries got changed from 0xFFFFFFFF to 0x1.
- Removed ray stream API as rarely used with minimal performance benefits over packet tracing.
- Introduced rtcForwardIntersect/rtcForwardOccluded API calls to trace tail recursive rays from user geometry callback.
- The rtcGetGeometryUserDataFromScene API call got added to be used in SYCL code.
- Added support for user geometry callback function pointer passed through ray query context

- Feature flags enable reducing code complexity for optimal performance on the GPU.
- Fixed compilation issues for ARM AArch64 processor under Linux.
- Setting default frequency level to SIMD256 for ARM on all platforms. This allows using double pumped NEON execution by enabling EMBREE_ISA_NEON2X in cmake under Linux.
- Fixed missing end caps of motion blurred line segments.
- EMBREE_ISPC_SUPPORT is turned OFF by default.
- Embree drops support of the deprecated Intel(R) Compiler. It is replaced by the Intel(R) oneAPI DPC++/C++ Compiler on Windows and Linux and the Intel(R) C++ Classic Compiler on MacOS (latest tested versions is 2023.0.0).

1.3.10 Embree 3.13.5

- Fixed bug in bounding flat Catmull Rom curves of subdivision level 4.
- Improved self intersection avoidance for RTC_GEOMETRY_TYPE_DISC_POINT geometry type. Intersections are skipped if the ray origin lies inside the sphere defined by the point primitive. Self intersection avoidance can get disabled at compile time using the EMBREE_DISC_POINT_SELF_INTERSECTION_AVOIDANCE cmake option.
- Fixed spatial splitting for non-planar quads.

1.3.11 Embree 3.13.4

- Using 8-wide BVH and double pumped NEON instructions on Apple M1 gives 8% performance boost.
- Fixed binning related crash in SAH BVH builder.
- Added EMBREE_TBB_COMPONENT cmake option to define the component/library name of Intel® TBB (default: tbb).
- Embree supports now Intel® oneAPI DPC++/C++ Compiler 2022.0.0

1.3.12 Embree 3.13.3

- Invalid multi segment motion blurred normal oriented curves are properly excluded from BVH build.
- Fixing issue with normal oriented curve construction when center curve curvature is very large. Due to this change normal oriented curve shape changes slightly.
- Fixed crash caused by disabling a geometry and then detaching it from the scene.
- Bugfix in emulated ray packet intersection when EMBREE_RAY_PACKETS is turned off.
- Bugfix for linear quaternion interpolation fallback.
- Fixed issues with spaces in path to Embree build folder.
- Some fixes to compile Embree in SSE mode using WebAssembly.
- Bugfix for occlusion rays with grids and ray packets.
- We do no longer provide installers for Windows and macOS, please use the ZIP files instead.
- Upgrading to Intel® ISPC 1.17.0 for release build.
- Upgrading to Intel® oneTBB 2021.5.0 for release build.

1.3.13 Embree 3.13.2

- Avoiding spatial split positions that are slightly out of geometry bounds.
- Introduced rtcGetGeometryThreadSafe function, which is a thread safe version of rtcGetGeometry.
- Using more accurate rcp implementation.

- Bugfix to rare corner case of high quality BVH builder.

1.3.14 Embree 3.13.1

- Added support for Intel® ISPC ARM target.
- Releases upgrade to Intel® TBB 2021.3.0 and Intel® ISPC 1.16.1

1.3.15 Embree 3.13.0

- Added support for Apple M1 CPUs.
- RTC_SUBDIVISION_MODE_NO_BOUNDARY now works properly for non-manifold edges.
- CMake target 'uninstall' is not defined if it already exists.
- Embree no longer reads the .embree3 config files, thus all configuration has to get passed through the config string to rtcNewDevice.
- Releases upgrade to Intel® TBB 2021.2.0 and Intel® ISPC 1.15.0
- Intel® TBB dll is automatically copied into build folder after build on windows.

1.3.16 Embree 3.12.2

- Fixed wrong uv and Ng for grid intersector in robust mode for AVX.
- Removed optimizations for Knights Landing.
- Upgrading release builds to use Intel® oneTBB 2021.1.1

1.3.17 Embree 3.12.1

- Changed default frequency level to SIMD128 for Skylake, Cannon Lake, Comet Lake and Tiger Lake CPUs. This change typically improves performance for renderers that just use SSE by maintaining higher CPU frequencies. In case your renderer is AVX optimized you can get higher ray tracing performance by configuring the frequency level to simd256 through passing frequency_level=simd256 to rtcNewDevice.

1.3.18 Embree 3.12.0

- Added linear cone curve geometry support. In this mode a real geometric surface for curves with linear basis is rendered using capped cones. They are discontinuous at edge boundaries.
- Enabled fast two level builder for instances when low quality build is requested.
- Bugfix for BVH build when geometries got disabled.
- Added EMBREE_BACKFACE_CULLING_CURVES cmake option. This allows for a cheaper round linear curve intersection when correct internal tracking and back hits are not required. The new cmake option defaults to OFF.
- User geometries with invalid bounds with lower>upper in some dimension will be ignored.
- Increased robustness for grid interpolation code and fixed returned out of range u/v coordinates for grid primitive.
- Fixed handling of motion blur time range for sphere, discs, and oriented disc geometries.
- Fixed missing model data in releases.
- Ensure compatibility to newer versions of Intel® oneTBB.
- Motion blur BVH nodes no longer store NaN values.

1.3.19 Embree 3.11.0

- Round linear curves now automatically check for the existence of left and right connected segments if the flags buffer is empty. Left segments exist if the segment(id-1) + 1 == segment(id) and similarly for right segments.
- Implemented the min-width feature for curves and points, which allows to increase the radius in a distance dependent way, such that the curve or points thickness is n pixels wide.
- Round linear curves are closed now also at their start.
- Embree no longer supports Visual Studio 2013 starting with this release.
- Bugfix in subdivision tessellation level assignment for non-quad base primitives
- Small meshes are directly added to top level build phase of two-level builder to reduce memory consumption.
- Enabled fast two level builder for user geometries when low quality build is requested.

1.3.20 Embree 3.10.0

- Added EMBREE_COMPACT_POLYS CMake option which enables double indexed triangle and quad leaves to reduce memory consumption in compact mode by an additional 40% at about 15% performance impact. This new mode is disabled by default.
- Compile fix for Intel® oneTBB 2021.1-beta05
- Releases upgrade to Intel® TBB 2020.2
- Compile fix for Intel® ISPC v1.13.0
- Adding RPATH to libembree.so in releases
- Increased required CMake version to 3.1.0
- Made instID member for array of pointers ray stream layout optional again.

1.3.21 Embree 3.9.0

- Added round linear curve geometry support. In this mode a real geometric surface for curves with linear basis is rendered using capped cones with spherical filling between the curve segments.
- Added rtcGetSceneDevice API function, that returns the device a scene got created in.
- Improved performance of round curve rendering by up to 1.8x.
- Bugfix to sphere intersection filter invocation for back hit.
- Fixed wrong assertion that triggered for invalid curves which anyway get filtered out.
- RelWithDebInfo mode no longer enables assertions.
- Fixed an issue in FindTBB.cmake that caused compile error with Debug build under Linux.
- Embree releases no longer provide RPMs for Linux. Please use the RPMs coming with the package manager of your Linux distribution.

1.3.22 Embree 3.8.0

- Added collision detection support for user geometries (see rtcCollide API function)
- Passing geomID to user geometry callbacks.
- Bugfix in AVX512VL codepath for rtcIntersect1
- For sphere geometries the intersection filter gets now invoked for front and back hit.
- Fixed some bugs for quaternion motion blur.
- RTCRayQueryContext always non-const in Embree API

- Made RTCHit aligned to 16 bytes in Embree API

1.3.23 New Features in Embree 3.7.0

- Added quaternion motion blur for correct interpolation of rotational transformations.
- Fixed wrong bounding calculations when a motion blurred instance did instantiate a motion blurred scene.
- In robust mode the depth test consistently uses tnear <= t <= tfar now in order to robustly continue traversal at a previous hit point in a way that guarantees reaching all hits, even hits at the same place.
- Fixed depth test in robust mode to be precise at tnear and tfar.
- Added next_hit tutorial to demonstrate robustly collecting all hits along a ray using multiple ray queries.
- Implemented robust mode for curves. This has a small performance impact but fixes bounding problems with flat curves.
- Improved quality of motion blur BVH by using linear bounds during binning.
- Implemented issue with motion blur builder where number of time segments for SAH heuristic were counted wrong due to some numerical issues.
- Fixed an accuracy issue with rendering very short fat curves.
- rtcCommitScene can now get called during rendering from multiple threads to lazily build geometry. When Intel® TBB is used this causes a much lower overhead than using rtcJoinCommitScene.
- Geometries can now get attached to multiple scenes at the same time, which simplifies mapping general scene graphs to API.
- Updated to Intel® TBB 2019.9 for release builds.
- Fixed a bug in the BVH builder for Grid geometries.
- Added macOS Catalina support to Embree releases.

1.3.24 New Features in Embree 3.6.1

- Restored binary compatibility between Embree 3.6 and 3.5 when single-level instancing is used.
- Fixed bug in subgrid intersector
- Removed point query alignment in Intel® ISPC header

1.3.25 New Features in Embree 3.6

- Added Catmull-Rom curve types.
- Added support for multi-level instancing.
- Added support for point queries.
- Fixed a bug preventing normal oriented curves being used unless timesteps were specified.
- Fixed bug in external BVH builder when configured for dynamic build.
- Added support for new config flag “user_threads=N” to device initialization which sets the number of threads used by Intel® TBB but created by the user.
- Fixed automatic vertex buffer padding when using rtcSetNewGeometry API function.

1.3.26 New Features in Embree 3.5.2

- Added EMBREE_API_NAMESPACE cmake option that allows to put all Embree API functions inside a user defined namespace.

- Added EMBREE_LIBRARY_NAME cmake option that allows to rename the Embree library.
- When Embree is compiled as static library, EMBREE_STATIC_LIB has no longer to get defined before including the Embree API headers.
- Added CPU frequency_level device configuration to allow an application to specify the frequency level it wants to run on. This forces Embree to not use optimizations that may reduce the CPU frequency below that level. By default Embree is configured to the the AVX-heavy frequency level, thus if the application uses solely non-AVX code, configuring the Embree device with "frequency_level=simd128" may give better performance.
- Fixed a bug in the spatial split builder which caused it to fail for scenes with more than 2^24 geometries.

1.3.27 New Features in Embree 3.5.1

- Fixed ray/sphere intersector to work also for non-normalized rays.
- Fixed self intersection avoidance for ray oriented discs when non-normalized rays were used.
- Increased maximal face valence for subdiv patch to 64 and reduced stack size requirement for subdiv patch evaluation.

1.3.28 New Features in Embree 3.5.0

- Changed normal oriented curve definition to fix waving artefacts.
- Fixed bounding issue for normal oriented motion blurred curves.
- Fixed performance issue with motion blurred point geometry.
- Fixed generation of documentation with new pandoc versions.

1.3.29 New Features in Embree 3.4.0

- Added point primitives (spheres, ray-oriented discs, normal-oriented discs).
- Fixed crash triggered by scenes with only invalid primitives.
- Improved robustness of quad/grid-based intersectors.
- Upgraded to Intel® TBB 2019.2 for release builds.

1.3.30 New Features in Embree 3.3.0

- Added support for motion blur time range per geometry. This way geometries can appear and disappear during the camera shutter and time steps do not have to start and end at camera shutter interval boundaries.
- Fixed crash with pathtracer when using –triangle-sphere command line.
- Fixed crash with pathtracer when using –shader ao command line.
- Fixed tutorials showing a black window on macOS 10.14 until moved.

1.3.31 New Features in Embree 3.2.4

- Fixed compile issues with ICC 2019.
- Released ZIP files for Windows are now provided in a version linked against Visual Studio 2013 and Visual Studio 2015.

1.3.32 New Features in Embree 3.2.3

- Fixed crash when using curves with RTC_SCENE_FLAG_DYNAMIC combined with RTC_BUILD_QUALITY_MEDIUM.

1.3.33 New Features in Embree 3.2.2

- Fixed intersection distance for unnormalized rays with line segments.
- Removed libmmd.dll dependency in release builds for Windows.
- Fixed detection of AppleClang compiler under MacOSX.

1.3.34 New Features in Embree 3.2.1

- Bugfix in flat mode for hermite curves.
- Added EMBREE_CURVE_SELF_INTERSECTION_AVOIDANCE_FACTOR cmake option to control self intersection avoidance for flat curves.
- Performance fix when instantiating motion blurred scenes. The application should best use two (or more) time steps for an instance that instantiates a motion blurred scene.
- Fixed AVX512 compile issue with GCC 6.1.1.
- Fixed performance issue with rtcGetGeometryUserData when used during rendering.
- Bugfix in length of derivatives for grid geometry.
- Added BVH8 support for motion blurred curves and lines. For some workloads this increases performance by up to 7%.
- Fixed rtcGetGeometryTransform to return the local to world transform.
- Fixed bug in multi segment motion blur that caused missing of perfectly axis aligned geometry.
- Reduced memory consumption of small scenes by 4x.
- Reduced temporal storage of grid builder.

1.3.35 New Features in Embree 3.2.0

- Improved watertightness of robust mode.
- Line segments, and other curves are now all contained in a single BVH which improves performance when these are both used in a scene.
- Performance improvement of up to 20% for line segments.
- Bugfix to Embree2 to Embree3 conversion script.
- Added support for Hermite curve basis.
- Semantics of normal buffer for normal oriented curves has changed to simplify usage. Please see documentation for details.
- Using GLFW and imgui in tutorials.
- Fixed floating point exception in static variable initialization.
- Fixed invalid memory access in rtcGetGeometryTransform for non-motion blur instances.
- Improved self intersection avoidance for flat curves. Transparency rays with tnear set to previous hit distance do not need curve radius based self intersection avoidance as same hit is calculated again. For this reason self intersection avoidance is now only applied to ray origin.

1.3.36 New Features in Embree 3.1.0

- Added new normal-oriented curve primitive for ray tracing of grass-like structures.
- Added new grid primitive for ray tracing tessellated and displaced surfaces in very memory efficient manner.
- Fixed bug of ribbon curve intersector when derivative was zero.
- Installing all static libraries when EMBREE_STATIC_LIB is enabled.
- Added API functions to access topology of subdivision mesh.
- Reduced memory consumption of instances.
- Improved performance of instances by 8%.
- Reduced memory consumption of curves by up to 2x.

- Up to 5% higher performance on AVX-512 architectures.
- Added native support for multiple curve basis functions. Internal basis conversions are no longer performed, which saves additional memory when multiple bases are used.
- Fixed issue with non thread safe local static variable initialization in VS2013.
- Bugfix in rtcSetNewGeometry. Vertex buffers did not get properly overallocated.
- Replaced ImageMagick with OpenImageIO in the tutorials.

1.3.37 New Features in Embree 3.0.0

- Switched to a new version of the API which provides improved flexibility but is not backward compatible. Please see "Upgrading from Embree 2 to Embree 3" section of the documentation for upgrade instructions. In particular, we provide a Python script that performs most of the transition work.
- User geometries inside an instanced scene and a top-level scene no longer need to handle the instID field of the ray differently. They both just need to copy the context.instID into the ray.instID field.
- Support for context filter functions that can be assigned to a ray query.
- User geometries can now invoke filter functions using the rtcFilterIntersection and rtcFilterOcclusion calls.
- Higher flexibility through specifying build quality per scene and geometry.
- Geometry normal uses commonly used right-hand rule from now on.
- Added self-intersection avoidance to ribbon curves and lines. Applications do not have to implement self-intersection workarounds for these primitive types anymore.
- Added support for 4 billion primitives in a single scene.
- Removed the RTC_MAX_USER_VERTEX_BUFFERS and RTC_MAX_INDEX_BUFFERS limitations.
- Reduced memory consumption by 192 bytes per instance.
- Fixed some performance issues on AVX-512 architectures.
- Individual Contributor License Agreement (ICLA) and Corporate Contributor License Agreement (CCLA) no longer required to contribute to the project.

1.3.38 New Features in Embree 2.17.5

- Improved watertightness of robust mode.
- Fixed floating point exception in static variable initialization.
- Fixed AVX512 compile issue with GCC 6.1.1.

1.3.39 New Features in Embree 2.17.4

- Fixed AVX512 compile issue with GCC 7.
- Fixed issue with not thread safe local static variable initialization in VS2013.
- Fixed bug in the 4 and 8-wide packet intersection of instances with multi-segment motion blur on AVX-512 architectures.
- Fixed bug in rtcOccluded4/8/16 when only AVX-512 ISA was enabled.

1.3.40 New Features in Embree 2.17.3

- Fixed GCC compile warning in debug mode.
- Fixed bug of ribbon curve intersector when derivative was zero.
- Installing all static libraries when EMBREE_STATIC_LIB is enabled.

1.3.41 New Features in Embree 2.17.2

- Made BVH build of curve geometry deterministic.

1.3.42 New Features in Embree 2.17.1

- Improved performance of occlusion ray packets by up to 50%.
- Fixed detection of Clang for CMake 3 under MacOSX
- Fixed AVX code compilation issue with GCC 7 compiler caused by explicit use of vzeroupper intrinsics.
- Fixed an issue where Clang address sanitizer reported an error in the internal tasking system.
- Added fix to compile on 32 bit Linux distribution.
- Fixed some wrong relative include paths in Embree.
- Improved performance of robust single ray mode by 5%.
- Added EMBREE_INSTALL_DEPENDENCIES option (default OFF) to enable installing of Embree dependencies.
- Fixed performance regression for occlusion ray streams.
- Reduced temporary memory requirements of BVH builder for curves and line segments.
- Fixed performance regression for user geometries and packet ray tracing.
- Fixed bug where wrong closest hit was reported for very curvy hair segment.

1.3.43 New Features in Embree 2.17.0

- Improved packet ray tracing performance for coherent rays by 10-60% (requires RTC_INTERSECT_COHERENT flag).
- Improved ray tracing performance for incoherent rays on AVX-512 architectures by 5%.
- Improved ray tracing performance for streams of incoherent rays by 5-15%.
- Fixed tbb_debug.lib linking error under Windows.
- Fast coherent ray stream and packet code paths now also work in robust mode.
- Using less aggressive prefetching for large BVH nodes which results in 1-2% higher ray tracing performance.
- Precompiled binaries have stack-protector enabled, except for traversal kernels. BVH builders can be slightly slower due to this change. If you want stack-protectors disabled please turn off EMBREE_STACK_PROTECTOR in cmake and build the binaries yourself.
- When enabling ISAs individually, the 8-wide BVH was previously only available when the AVX ISA was also selected. This issue is now fixed, and one can enable only AVX2 and still get best performance by using an 8-wide BVH.
- Fixed rtcOccluded1 and rtcOccluded1Ex API functions which were broken in Intel® ISPC.
- Providing MSI installer for Windows.

1.3.44 New Features in Embree 2.16.5

- Bugfix in the robust triangle intersector that rarely caused NaNs.
- Fixed bug in hybrid traversal kernel when BVH leaf was entered with no active rays. This rarely caused crashes when used with instancing.
- Fixed bug introduced in Embree 2.16.2 which caused instancing not to work properly when a smaller than the native SIMD width was used in ray packet mode.

- Fixed bug in the curve geometry intersector that caused rendering artefacts for Bézier curves with p0=p1 and/or p2=p3.
- Fixed bug in the curve geometry intersector that caused hit results with NaNs to be reported.
- Fixed masking bug that caused rare cracks in curve geometry.
- Enabled support for SSE2 in precompiled binaries again.

1.3.45 New Features in Embree 2.16.4

- Bugfix in the ribbon intersector for hair primitives. Non-normalized rays caused wrong intersection distance to be reported.

1.3.46 New Features in Embree 2.16.3

- Increased accuracy for handling subdivision surfaces. This fixes cracks when using displacement mapping but reduces performance at irregular vertices.
- Fixed a bug where subdivision geometry was not properly updated when modifying only the tessellation rate and vertex array.

1.3.47 New Features in Embree 2.16.2

- Fixed bug that caused NULL ray query context in intersection filter when instancing was used.
- Fixed an issue where uv's where outside the triangle (or quad) for very small triangles (or quads). In robust mode we improved the uv calculation to avoid that issue, in fast mode we accept that inconsistency for better performance.
- Changed UV encoding for non-quad subdivision patches to allow a sub-patch UV range of `[-0.5,1.5[`. Using this new encoding one can use finite differences to calculate derivatives if required. Please adjust your code in case you rely on the old encoding.

1.3.48 New Features in Embree 2.16.1

- Workaround for compile issues with Visual Studio 2017
- Fixed bug in subdiv code for static scenes when using tessellation levels larger than 50.
- Fixed low performance when adding many geometries to a scene.
- Fixed high memory consumption issue when using instances in dynamic scene (by disabling two level builder for user geometries and instances).

1.3.49 New Features in Embree 2.16.0

- Improved multi-segment motion blur support for scenes with different number of time steps per mesh.
- New top level BVH builder that improves build times and BVH quality of two-level BVHs.
- Added support to enable only a single ISA. Previously code was always compiled for SSE2.
- Improved single ray tracing performance for incoherent rays on AVX-512 architectures by 5-10%.
- Improved packet/hybrid ray tracing performance for incoherent rays on AVX-512 architectures by 10-30%.
- Improved stream ray tracing performance for coherent rays in structure-of-pointers layout by 40-70%.

- BVH builder for compact scenes of triangles and quads needs essentially no temporary memory anymore. This doubles the maximal scene size that can be rendered in compact mode.
- Triangles no longer store the geometry normal in fast/default mode which reduces memory consumption by up to 20%.
- Compact mode uses BVH4 now consistently which reduces memory consumption by up to 10%.
- Reduced memory consumption for small scenes (of 10k-100k primitives) and dynamic scenes.
- Improved performance of user geometries and instances through BVH8 support.
- The API supports now specifying the geometry ID of a geometry at construction time. This way matching the geometry ID used by Embree and the application is simplified.
- Fixed a bug that would have caused a failure of the BVH builder for dynamic scenes when run on a machine with more then 1000 threads.
- Fixed a bug that could have been triggered when reaching the maximal number of mappings under Linux (`vm.max_map_count`). This could have happened when creating a large number of small static scenes.
- Added huge page support for Windows and MacOSX (experimental).
- Added support for Visual Studio 2017.
- Removed support for Visual Studio 2012.
- Precompiled binaries now require a CPU supporting at least the SSE4.2 ISA.
- We no longer provide precompiled binaries for 32-bit on Windows.
- Under Windows one now has to use the platform toolset option in CMake to switch to Clang or the Intel® Compiler.
- Fixed a bug for subdivision meshes when using the incoherent scene flag.
- Fixed a bug in the line geometry intersection, that caused reporting an invalid line segment intersection with primID -1.
- Buffer stride for vertex buffers of different time steps of triangle and quad meshes have to be identical now.
- Fixed a bug in the curve geometry intersection code when passed a perfect cylinder.

1.3.50 New Features in Embree 2.15.0

- Added `rtcCommitJoin` mode that allows thread to join a build operation. When using the internal tasking system this allows Embree to solely use the threads that called `rtcCommitJoin` to build the scene, while previously also normal worker threads participated in the build. You should no longer use `rtcCommit` to join a build.
- Added `rtcDeviceSetErrorFunction2` API call, which sets an error callback function which additionally gets passed a user provided pointer (`rtcDeviceSetErrorFunction` is now deprecated).
- Added `rtcDeviceSetMemoryMonitorFunction2` API call, which sets a memory monitor callback function which additionally get passed a user provided pointer. (`rtcDeviceSetMemoryMonitorFunction` is now deprecated).
- Build performance for hair geometry improved by up to 2×.
- Standard BVH build performance increased by 5%.
- Added API extension to use internal Morton-code based builder, the standard binned-SAH builder, and the spatial split-based SAH builder.
- Added support for BSpline hair and curves. Embree uses either the Bézier or BSpline basis internally, and converts other curves, which requires more memory during rendering. For reduced memory consumption set the `EM-`

BREE_NATIVE_SPLINE_BASIS to the basis your application uses (which is set to BEZIER by default).
- Setting the number of threads through tbb::taskscheduler_init object on the application side is now working properly.
- Windows and Linux releases are build using AVX-512 support.
- Implemented hybrid traversal for hair and line segments for improved ray packet performance.
- AVX-512 code compiles with Clang 4.0.0
- Fixed crash when ray packets were disabled in CMake.

1.3.51 New Features in Embree 2.14.0

- Added ignore_config_files option to init flags that allows the application to ignore Embree configuration files.
- Face-varying interpolation is now supported for subdivision surfaces.
- Up to 16 user vertex buffers are supported for vertex attribute interpolation.
- Deprecated rtcSetBoundaryMode function, please use the new rtcSetSubdivisionMode function.
- Added RTC_SUBDIV_PIN_BOUNDARY mode for handling boundaries of subdivision meshes.
- Added RTC_SUBDIV_PIN_ALL mode to enforce linear interpolation for subdivision meshes.
- Optimized object generation performance for dynamic scenes.
- Reduced memory consumption when using lots of small dynamic objects.
- Fixed bug for subdivision surfaces using low tessellation rates.
- Hair geometry now uses a new ribbon intersector that intersects with ray-facing quads. The new intersector also returns the v-coordinate of the hair intersection, and fixes artefacts at junction points between segments, at the cost of a small performance hit.
- Added rtcSetBuffer2 function, that additionally gets the number of elements of a buffer. In dynamic scenes, this function allows to quickly change buffer sizes, making it possible to change the number of primitives of a mesh or the number of crease features for subdivision surfaces.
- Added simple 'viewer_anim' tutorial for rendering key frame animations and 'buildbench' for measuring BVH (re-)build performance for static and dynamic scenes.
- Added more AVX-512 optimizations for future architectures.

1.3.52 New Features in Embree 2.13.0

- Improved performance for compact (but not robust) scenes.
- Added robust mode for motion blurred triangles and quads.
- Added fast dynamic mode for user geometries.
- Up to 20% faster BVH build performance on the second generation Intel® Xeon Phi™ processor codenamed Knights Landing.
- Improved quality of the spatial split builder.
- Improved performance for coherent streams of ray packets (SOA layout), e.g. for fast primary visibility.
- Various bug fixes in tessellation cache, quad-based spatial split builder, etc.

1.3.53 New Features in Embree 2.12.0

- Added support for multi-segment motion blur for all primitive types.
- API support for stream of pointers to single rays (rtcIntersect1Mp and rtcOccluded1Mp)

- Improved BVH refitting performance for dynamic scenes.
- Improved high-quality mode for quads (added spatial split builder for quads)
- Faster dynamic scenes for triangle and quad-based meshes on AVX2 enabled machines.
- Performance and correctness bugfix in optimization for streams of coherent (single) rays.
- Fixed large memory consumption (issue introduced in Embree v2.11.0). If you use Embree v2.11.0 please upgrade to Embree v2.12.0.
- Reduced memory consumption for dynamic scenes containing small meshes.
- Added support to start and affinitize Intel® TBB worker threads by passing "`start_threads=1,set_affinity=1`" to `rtcNewDevice`. These settings are recommended on systems with a high thread count.
- `rtcInterpolate2` can now be called within a displacement shader.
- Added initial support for Microsoft's Parallel Pattern Library (PPL) as tasking system alternative (for optimal performance Intel® TBB is highly recommended).
- Updated to Intel® TBB 2017 which is released under the Apache v2.0 license.
- Dropped support for Visual Studio 2012 Win32 compiler. Visual Studio 2012 x64 is still supported.

1.3.54 New Features in Embree 2.11.0

- Improved performance for streams of coherent (single) rays flagged with `RTC_INTERSECT_COHERENT`. For such coherent ray streams, e.g. primary rays, the performance typically improves by 1.3-2×.
- New spatial split BVH builder for triangles, which is 2-6× faster than the previous version and more memory conservative.
- Improved performance and scalability of all standard BVH builders on systems with large core counts.
- Fixed `rtcGetBounds` for motion blur scenes.
- Thread affinity is now on by default when running on the latest Intel® Xeon Phi™ processor.
- Added AVX-512 support for future Intel® Xeon processors.

1.3.55 New Features in Embree 2.10.0

- Added a new curve geometry which renders the sweep surface of a circle along a Bézier curve.
- Intersection filters can update the `tfar` ray distance.
- Geometry types can get disabled at compile time.
- Modified and extended the ray stream API.
- Added new callback mechanism for the ray stream API.
- Improved ray stream performance (up to 5-10%).
- Up to 20% faster morton builder on machines with large core counts.
- Lots of optimizations for the second generation Intel® Xeon Phi™ processor codenamed Knights Landing.
- Added experimental support for compressed BVH nodes (reduces node size to 56-62% of uncompressed size). Compression introduces a typical performance overhead of ~10%.
- Bugfix in backface culling mode. We do now properly cull the backfaces and not the frontfaces.
- Feature freeze for the first generation Intel® Xeon Phi™ coprocessor codenamed Knights Corner. We will still maintain and add bug fixes to Embree v2.9.0, but Embree 2.10 and future versions will no longer support it.

1.3.56 New Features in Embree 2.9.0

- Improved shadow ray performance (10-100% depending on the scene).
- Added initial support for ray streams (10-30% higher performance depending on ray coherence in the stream).
- Added support to calculate second order derivatives using the `rtcInterpolate2` function.
- Changed the parametrization for triangular subdivision faces to the same scheme used for pentagons.
- Added support to query the Embree configuration using the `rtcDeviceGetParameter` function.

1.3.57 New Features in Embree 2.8.1

- Added support for setting per geometry tessellation rate (supported for subdivision and Bézier geometries).
- Added support for motion blurred instances.

1.3.58 New Features in Embree 2.8.0

- Added support for line segment geometry.
- Added support for quad geometry (replaces triangle-pairs feature).
- Added support for linear motion blur of user geometries.
- Improved performance through AVX-512 optimizations.
- Improved performance of lazy scene build (when using Intel® TBB 4.4 update 2).
- Improved performance through huge page support under linux.

1.3.59 New Features in Embree 2.7.1

- Internal tasking system supports cancellation of build operations.
- Intel® ISPC mode for robust and compact scenes got significantly faster (implemented hybrid traversal for bvh4.triangle4v and bvh4.triangle4i).
- Hair rendering got faster as we fixed some issues with the SAH heuristic cost factors.
- BVH8 got slight faster for single ray traversal (improved sorting when hitting more than 4 boxes).
- BVH build performance got up to 30% faster on CPUs with high core counts (improved parallel partition code).
- High quality build mode again working properly (spatial splits had been deactivated in v2.7.0 due to some bug).
- Support for merging two adjacent triangles sharing a common edge into a triangle-pair primitive (can reduce memory consumption and BVH build times by up to 50% for mostly quad-based input meshes).
- Internal cleanups (reduced number of traversal kernels by more templating).
- Reduced stack size requirements of BVH builders.
- Fixed crash for dynamic scenes, triggered by deleting all geometries from the scene.

1.3.60 New Features in Embree 2.7.0

- Added device concept to Embree to allow different components of an application to use Embree without interfering with each other.
- Fixed memory leak in twolevel builder used for dynamic scenes.
- Fixed bug in tessellation cache that caused crashes for subdivision surfaces.

- Fixed bug in internal task scheduler that caused deadlocks when using `rtcCommitThread`.
- Improved hit-distance accuracy for thin triangles in robust mode.
- Added support to disable ray packet support in cmake.

1.3.61 New Features in Embree 2.6.2

- Fixed bug triggered by instantiating motion blur geometry.
- Fixed bug in hit UV coordinates of static subdivision geometries.
- Performance improvements when only changing tessellation levels for subdivision geometry per frame.
- Added ray packet intersectors for subdivision geometry, resulting in improved performance for coherent rays.
- Reduced virtual address space usage for static geometries.
- Fixed some AVX2 code paths when compiling with GCC or Clang.
- Bugfix for subdiv patches with non-matching winding order.
- Bugfix in ISA detection of AVX-512.

1.3.62 New Features in Embree 2.6.1

- Major performance improvements for ray tracing subdivision surfaces, e.g. up to 2× faster for scenes where only the tessellation levels are changing per frame, and up to 3× faster for scenes with lots of crease features
- Initial support for architectures supporting the new 16-wide AVX-512 ISA
- Implemented intersection filter callback support for subdivision surfaces
- Added `RTC_IGNORE_INVALID_RAYS` CMake option which makes the ray intersectors more robust against full tree traversal caused by invalid ray inputs (e.g. INF, NaN, etc)

1.3.63 New Features in Embree 2.6.0

- Added `rtcInterpolate` function to interpolate per vertex attributes
- Added `rtcSetBoundaryMode` function that can be used to select the boundary handling for subdivision surfaces
- Fixed a traversal bug that caused rays with very small ray direction components to miss geometry
- Performance improvements for the robust traversal mode
- Fixed deadlock when calling `rtcCommit` from multiple threads on same scene

1.3.64 New Features in Embree 2.5.1

- On dual socket workstations, the initial BVH build performance almost doubled through a better memory allocation scheme
- Reduced memory usage for subdivision surface objects with crease features
- `rtcCommit` performance is robust against unset “flush to zero” and “denormals are zero” flags. However, enabling these flags in your application is still recommended
- Reduced memory usage for subdivision surfaces with borders and infinitely sharp creases
- Lots of internal cleanups and bug fixes for both Intel® Xeon® and Intel® Xeon Phi™

1.3.65 New Features in Embree 2.5.0

- Improved hierarchy build performance on both Intel Xeon and Intel Xeon Phi
- Vastly improved tessellation cache for ray tracing subdivision surfaces
- Added `rtcGetUserData` API call to query per geometry user pointer set through `rtcSetUserData`
- Added support for memory monitor callback functions to track and limit memory consumption
- Added support for progress monitor callback functions to track build progress and cancel long build operations
- BVH builders can be used to build user defined hierarchies inside the application (see tutorial BVH Builder)
- Switched to Intel® TBB as default tasking system on Xeon to get even faster hierarchy build times and better integration for applications that also use Intel® TBB
- `rtcCommit` can get called from multiple Intel® TBB threads to join the hierarchy build operations

1.3.66 New Features in Embree 2.4

- Support for Catmull Clark subdivision surfaces (triangle/quad base primitives)
- Support for vector displacements on Catmull Clark subdivision surfaces
- Various bug fixes (e.g. 4-byte alignment of vertex buffers works)

1.3.67 New Features in Embree 2.3.3

- BVH builders more robustly handle invalid input data (Intel Xeon processor family)
- Motion blur support for hair geometry (Xeon)
- Improved motion blur performance for triangle geometry (Xeon)
- Improved robust ray tracing mode (Xeon)
- Added `rtcCommitThread` API call for easier integration into existing tasking systems (Xeon and Intel Xeon Phi coprocessor)
- Added support for recording and replaying all `rtcIntersect`/`rtcOccluded` calls (Xeon and Xeon Phi)

1.3.68 New Features in Embree 2.3.2

- Improved mixed AABB/OBB-BVH for hair geometry (Xeon Phi)
- Reduced amount of pre-allocated memory for BVH builders (Xeon Phi)
- New 64-bit Morton code-based BVH builder (Xeon Phi)
- (Enhanced) Morton code-based BVH builders use now tree rotations to improve BVH quality (Xeon Phi)
- Bug fixes (Xeon and Xeon Phi)

1.3.69 New Features in Embree 2.3.1

- High quality BVH mode improves spatial splits which result in up to 30% performance improvement for some scenes (Xeon)
- Compile time enabled intersection filter functions do not reduce performance if no intersection filter is used in the scene (Xeon and Xeon Phi)
- Improved ray tracing performance for hair geometry by >20% on Xeon Phi. BVH for hair geometry requires 20% less memory
- BVH8 for AVX/AVX2 targets improves performance for single ray tracing on Haswell by up to 12% and by up to 5% for hybrid (Xeon)

- Memory conservative BVH for Xeon Phi now uses BVH node quantization to lower memory footprint (requires half the memory footprint of the default BVH)

1.3.70 New Features in Embree 2.3

- Support for ray tracing hair geometry (Xeon and Xeon Phi)
- Catching errors through error callback function
- Faster hybrid traversal (Xeon and Xeon Phi)
- New memory conservative BVH for Xeon Phi
- Faster Morton code-based builder on Xeon
- Faster binned-SAH builder on Xeon Phi
- Lots of code cleanups/simplifications/improvements (Xeon and Xeon Phi)

1.3.71 New Features in Embree 2.2

- Support for motion blur on Xeon Phi
- Support for intersection filter callback functions
- Support for buffer sharing with the application
- Lots of AVX2 optimizations, e.g. ~20% faster 8-wide hybrid traversal
- Experimental support for 8-wide (AVX/AVX2) and 16-wide BVHs (Xeon Phi)

1.3.72 New Features in Embree 2.1

- New future proof API with a strong focus on supporting dynamic scenes
- Lots of optimizations for 8-wide AVX2 (Haswell architecture)
- Automatic runtime code selection for SSE, AVX, and AVX2
- Support for user-defined geometry
- New and improved BVH builders:
  - Fast adaptive Morton code-based builder (without SAH-based top-level rebuild)
  - Both the SAH and Morton code-based builders got faster (Xeon Phi)
  - New variant of the SAH-based builder using triangle pre-splits (Xeon Phi)

1.3.73 New Features in Embree 2.0

- Support for the Intel® Xeon Phi™ coprocessor platform
- Support for high-performance "packet" kernels on SSE, AVX, and Xeon Phi
- Integration with the Intel® Implicit SPMD Program Compiler (Intel® ISPC)
- Instantiation and fast BVH reconstruction
- Example photo-realistic rendering engine for both C++ and Intel® ISPC

Chapter 2

Installation of Embree

2.1 Windows Installation

A pre-built version of Embree for Windows is provided as a ZIP archive embree-4.4.0.x64.windows.zip. After unpacking this ZIP file, you should set the path to the `lib` folder manually to your `PATH` environment variable for applications to find Embree.

2.2 Linux Installation

A pre-built version of Embree for Linux is provided as a `tar.gz` archive: embree-4.4.0.x86_64.linux.tar.gz. Unpack this file using `tar` and source the provided `embree-vars.sh` (if you are using the bash shell) or `embree-vars.csh` (if you are using the C shell) to set up the environment properly:

```
tar xzf embree-4.4.0.x86_64.linux.tar.gz
source embree-4.4.0.x86_64.linux/embree-vars.sh
```

We recommend adding a relative `RPATH` to your application that points to the location where Embree (and TBB) can be found, e.g. `$ORIGIN/../lib`.

2.3 macOS Installation

The macOS version of Embree is also delivered as a ZIP file: embree-4.4.0.x86_64.macosx.zip. Unpack this file using `tar` and source the provided `embree-vars.sh` (if you are using the bash shell) or `embree-vars.csh` (if you are using the C shell) to set up the environment properly:

```
unzip embree-4.4.0.x64.macosx.zip    source embree-4.4.0.x64.macosx/embree-vars.sh
```

If you want to ship Embree with your application, please use the Embree library of the provided ZIP file. The library name of that Embree library is of the form `@rpath/libembree.4.dylib` (and similar also for the included TBB library). This ensures that you can add a relative `RPATH` to your application that points to the location where Embree (and TBB) can be found, e.g. `@loader_path/../lib`.

2.4 Building Embree Applications

The most convenient way to build an Embree application is through CMake. Just let CMake find your unpacked Embree package using the `FIND_PACKAGE` function inside your `CMakeLists.txt` file:

```
FIND_PACKAGE(embree 4 REQUIRED)
```

For CMake to properly find Embree you need to set the `embree_DIR` variable to the folder containing the `embree_config.cmake` file. You might also have to set the `TBB_DIR` variable to the path containing `TBB-config.cmake` of a local TBB install, in case you do not have TBB installed globally on your system, e.g:

```
cmake -D embree_DIR=path_to_embree_package/lib/cmake/embree-4.4.0/ \
      -D TBB_DIR=path_to_tbb_package/lib/cmake/tbb/ \
      ..
```

The `FIND_PACKAGE` function will create an `embree` target that you can add to your target link libraries:

```
TARGET_LINK_LIBRARIES(application embree)
```

For a full example on how to build an Embree application please have a look at the `minimal` tutorial provided in the `src` folder of the Embree package and also the contained `README.txt` file.

2.5 Building Embree SYCL Applications

Building Embree SYCL applications is also best done using CMake. Please first get some compatible SYCL compiler and setup the environment as decribed in sections Linux SYCL Compilation and Windows SYCL Compilation.

Also perform the setup steps from the previous Building Embree Applications section.

Please also have a look at the Minimal tutorial that is provided with the Embree release, for an example how to build a simple SYCL application using CMake and Embree.

To properly compile your SYCL application you have to add additional SYCL compile flags for each C++ file that contains SYCL device side code or kernels as described next.

2.5.1 JIT Compilation

We recommend using just in time compilation (JIT compilation) together with [SYCL JIT caching] to compile Embree SYCL applications. For JIT compilation add these options to the compilation phase of all C++ files that contain SYCL code:

```
-fsycl -Xclang -fsycl-allow-func-ptr -fsycl-targets=spir64
```

These options enable SYCL two phase compilation (`-fsycl` option), enable function pointer support (`-Xclang -fsycl-allow-func-ptr` option), and just in time (JIT) compilation only (`-fsycl-targets=spir64` option).

The following link options have to get added to the linking stage of your application when using just in time compilation:

```
-fsycl -fsycl-targets=spir64
```

For a full example on how to build an Embree SYCL application please have a look at the SYCL version of the `minimal` tutorial provided in the `src` folder of the Embree package and also the contained `README.txt` file.

Please have a look at the Compiling Embree section on how to create an Embree package from sources if required.

2.5.2 AOT Compilation

Ahead of time compilation (AOT compilation) allows to speed up first application start up time as device binaries are precompiled. We do not recommend using AOT compilation as it does not allow the usage of specialization constants to reduce code complexity.

For ahead of time compilation add these compile options to the compilation phase of all C++ files that contain SYCL code:

```
-fsycl -Xclang -fsycl-allow-func-ptr -fsycl-targets=spir64_gen
```

These options enable SYCL two phase compilation (`-fsycl` option), enable function pointer support (`-Xclang -fsycl-allow-func-ptr` option), and ahead of time (AOT) compilation (`-fsycl-targets=spir64_gen` option).

The following link options have to get added to the linking stage of your application when compiling ahead of time for Xe HPG devices:

```
-fsycl -fsycl-targets=spir64_gen
-Xsycl-target-backend=spir64_gen "-device XE_HPG_CORE"
```

This in particular configures the devices for AOT compilation to `XE_HPG_CORE`.

To get a list of all device supported by AOT compilation look at the help of the device option in ocloc tool:

```
ocloc compile --help
```

2.6 Building Embree Tests

Embree is released with a bundle of tests in an optional testing package. To run these tests extract the testing package in the same folder as your embree installation. e.g.:

```
tar -xzf embree-4.4.0-testing.zip -C /path/to/installed/embree
```

The tests are extracted into a new folder inside you embree installation and can be run with:

```
cd /path/to/installed/embree/testing
cmake -B build
cmake --build build target=tests
```

Chapter 3
Compiling Embree

We recommend using the prebuild Embree packages from https://github.com/embree/embree/releases. If you need to compile Embree yourself you need to use CMake as described in the following.

Do not enable fast-math optimizations in your compiler as this mode is not supported by Embree.

3.1 Linux and macOS

To compile Embree you need a modern C++ compiler that supports C++11. Embree is tested with the following compilers:

Linux

- Intel® oneAPI DPC++/C++ Compiler 2024.0.2
- oneAPI DPC++/C++ Compiler 2023-10-26
- Clang 5.0.0
- Clang 4.0.0
- GCC 10.0.1 (Fedora 32) AVX512 support
- GCC 8.3.1 (Fedora 29) AVX512 support
- Intel® Implicit SPMD Program Compiler 1.22.0

macOS x86_64

- Apple Clang 15

macOS Arm64

- Apple Clang 14

Embree supports using the Intel® Threading Building Blocks (TBB) as the tasking system. For performance and flexibility reasons we recommend using Embree with the Intel® Threading Building Blocks (TBB) and best also use TBB inside your application. Optionally you can disable TBB in Embree through the `EMBREE_TASKING_SYSTEM` CMake variable.

Embree supports the Intel® Implicit SPMD Program Compiler (Intel® ISPC), which allows straightforward parallelization of an entire renderer. If you want to use Intel® ISPC then you can enable `EMBREE_ISPC_SUPPORT` in CMake. Download and install the Intel® ISPC binaries from ispc.github.io. After installation, put the path to `ispc` permanently into your `PATH` environment variable or you set the `EMBREE_ISPC_EXECUTABLE` variable to point at the ISPC executable during CMake configuration.

You additionally have to install CMake 3.1.0 or higher and the developer version of GLFW version 3.

Under macOS, all these dependencies can be installed using MacPorts:

```
sudo port install cmake tbb glfw-devel
```

Depending on your Linux distribution you can install these dependencies using `yum` or `apt-get`. Some of these packages might already be installed or might have slightly different names.

Type the following to install the dependencies using `yum`:

```
sudo yum install cmake
sudo yum install tbb-devel
sudo yum install glfw-devel
```

Type the following to install the dependencies using `apt-get`:

```
sudo apt-get install cmake-curses-gui
sudo apt-get install libtbb-dev
sudo apt-get install libglfw3-dev
```

Finally, you can compile Embree using CMake. Create a build directory inside the Embree root directory and execute `ccmake ..` inside this build directory.

```
mkdir build
cd build
ccmake ..
```

Per default, CMake will use the compilers specified with the `CC` and `CXX` environment variables. Should you want to use a different compiler, run `cmake` first and set the `CMAKE_CXX_COMPILER` and `CMAKE_C_COMPILER` variables to the desired compiler. For example, to use the Clang compiler instead of the default GCC on most Linux machines (`g++` and `gcc`), execute

```
cmake -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_C_COMPILER=clang ..
```

Running `ccmake` will open a dialog where you can perform various configurations as described below in CMake Configuration. After having configured Embree, press `c` (for configure) and `g` (for generate) to generate a Makefile and leave the configuration. The code can be compiled by executing make.

```
make -j 8
```

The executables will be generated inside the build folder. We recommend installing the Embree library and header files on your system. Therefore set the `CMAKE_INSTALL_PREFIX` to `/usr` in cmake and type:

```
sudo make install
```

If you keep the default `CMAKE_INSTALL_PREFIX` of `/usr/local` then you have to make sure the path `/usr/local/lib` is in your `LD_LIBRARY_PATH`.

You can also uninstall Embree again by executing:

```
sudo make uninstall
```

You can also create an Embree package using the following command:

```
make package
```

Please see the Building Embree Applications section on how to build your application with such an Embree package.

3.2 Linux SYCL Compilation

There are two options to compile Embree with SYCL support: The open source "oneAPI DPC++ Compiler" or the "Intel(R) oneAPI DPC++/C++ Compiler". Other SYCL compilers are not supported.

The "oneAPI DPC++ Compiler" is more up-to-date than the "Intel(R) oneAPI DPC++/C++ Compiler" but less stable. The current tested version of the "oneAPI DPC++ compiler is

- oneAPI DPC++ Compiler 2023-10-26

The compiler can be downloaded and simply extracted. The oneAPI DPC++ compiler can be set up executing the following commands in a Linux (bash) shell:

```
export SYCL_BUNDLE_ROOT=path_to_dpcpp_compiler
export PATH=$SYCL_BUNDLE_ROOT/bin:$PATH
export CPATH=$SYCL_BUNDLE_ROOT/include:$CPATH
export LIBRARY_PATH=$SYCL_BUNDLE_ROOT/lib:$LIBRARY_PATH
export LD_LIBRARY_PATH=$SYCL_BUNDLE_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$SYCL_BUNDLE_ROOT/linux/lib/x64:$LD_LIBRARY_PATH
```

where the `path_to_dpcpp_compiler` should point to the unpacked oneAPI DPC++ compiler. This will put `clang++` and `clang` from the oneAPI DPC++ Compiler into your path.

Please also install all Linux packages described in the previous section.

Now, you can configure Embree using CMake by executing the following command in the Embree root directory:

```
cmake -B build \
      -DCMAKE_CXX_COMPILER=clang++ \
      -DCMAKE_C_COMPILER=clang \
      -DEMBREE_SYCL_SUPPORT=ON
```

This will create a directory `build` to use as the CMake build directory, configure the usage of the oneAPI DPC++ Compiler, and turn on SYCL support through `EMBREE_SYCL_SUPPORT=ON`.

Alternatively, you can download and run the installer of the

- Intel(R) oneAPI DPC++/C++ Compiler.

After installation, you can set up the compiler by sourcing the `vars.sh` script in the `env` directory of the compiler install directory, for example,

```
source /opt/intel/oneAPI/compiler/latest/env/vars.sh
```

This script will put the `icpx` and `icx` compiler executables from the Intel(R) oneAPI DPC++/C++ Compiler in your path.

Now, you can configure Embree using CMake by executing the following command in the Embree root directory:

```
cmake -B build \
      -DCMAKE_CXX_COMPILER=icpx \
      -DCMAKE_C_COMPILER=icx \
      -DEMBREE_SYCL_SUPPORT=ON
```

More information about setting up the Intel(R) oneAPI DPC++/C++ compiler can be found in the Development Reference Guide. Please note, that the Intel(R) oneAPI DPC++/C++ compiler requires at least CMake version 3.20.5 on Linux.

Independent of the DPC++ compiler choice, you can now build Embree using

```
cmake --build build -j 8
```

The executables will be generated inside the build folder. The executable names of the SYCL versions of the tutorials end with `_sycl`.

3.2.1 Linux Graphics Driver Installation

To run the SYCL code you need to install the latest GPGPU drivers for your Intel Xe HPG/HPC GPUs from here https://dgpu-docs.intel.com/. Follow the driver installation instructions for your graphics card and operating system.

After installing the drivers you have to install an additional package manually using

```
sudo apt install intel-level-zero-gpu-raytracing
```

3.3 Windows

Embree is tested using the following compilers under Windows:

- Intel® oneAPI DPC++/C++ Compiler 2024.0.2
- oneAPI DPC++/C++ Compiler 2023-10-26
- Visual Studio 2022
- Visual Studio 2019
- Visual Studio 2017
- Intel® Implicit SPMD Program Compiler 1.22.0

To compile Embree for AVX-512 you have to use the Intel® Compiler.

Embree supports using the Intel® Threading Building Blocks (TBB) as the tasking system. For performance and flexibility reasons we recommend using use Embree with the Intel® Threading Building Blocks (TBB) and best also use TBB inside your application. Optionally you can disable TBB in Embree through the `EMBREE_TASKING_SYSTEM` CMake variable.

Embree will either find the Intel® Threading Building Blocks (TBB) installation that comes with the Intel® Compiler, or you can install the binary distribution of TBB directly from https://github.com/oneapi-src/oneTBB/releases into a folder named `tbb` into your Embree root directory. You also have to make sure that the libraries `tbb.dll` and `tbb_malloc.dll` can be found when executing your Embree applications, e.g. by putting the path to these libraries into your `PATH` environment variable.

Embree supports the Intel® Implicit SPMD Program Compiler (Intel® ISPC), which allows straightforward parallelization of an entire renderer. When installing Intel® ISPC, make sure to download an Intel® ISPC version from ispc.github.io that is compatible with your Visual Studio version. After installation, put the path to `ispc.exe` permanently into your `PATH` environment variable or you need to correctly set the `EMBREE_ISPC_EXECUTABLE` variable during CMake configuration to point to the ISPC executable. If you want to use Intel® ISPC, you have to enable `EMBREE_ISPC_SUPPORT` in CMake.

You additionally have to install CMake (version 3.1 or higher). Note that you need a native Windows CMake installation because CMake under Cygwin cannot generate solution files for Visual Studio.

3.3.1 Using the IDE

Run `cmake-gui`, browse to the Embree sources, set the build directory and click Configure. Now you can select the Generator, e.g. "Visual Studio 12 2013" for a 32-bit build or "Visual Studio 12 2013 Win64" for a 64-bit build.

To use a different compiler than the Microsoft Visual C++ compiler, you additionally need to specify the proper compiler toolset through the option "Optional toolset to use (-T parameter)". E.g. to use Clang for compilation set the toolset to "LLVM_v142".

Do not change the toolset manually in a solution file (neither through the project properties dialog nor through the "Use Intel Compiler" project context

menu), because then some compiler-specific command line options cannot be set by CMake.

Most configuration parameters described in the CMake Configuration can be set under Windows as well. Finally, click "Generate" to create the Visual Studio solution files.

The following CMake options are only available under Windows:

- `CMAKE_CONFIGURATION_TYPE`: List of generated configurations. The default value is Debug;Release;RelWithDebInfo.
- `USE_STATIC_RUNTIME`: Use the static version of the C/C++ runtime library. This option is turned OFF by default.

Use the generated Visual Studio solution file `embree4.sln` to compile the project.

We recommend enabling syntax highlighting for the `.ispc` source and `.isph` header files. To do so open Visual Studio, go to Tools ⇒ Options ⇒ Text Editor ⇒ File Extension and add the `isph` and `ispc` extensions for the "Microsoft Visual C++" editor.

3.3.2 Using the Command Line

Embree can also be configured and built without the IDE using the Visual Studio command prompt:

```
cd path\to\embree
mkdir build
cd build
cmake -G "Visual Studio 16 2019" ..
cmake --build . --config Release
```

You can also build only some projects with the `--target` switch. Additional parameters after "`--`" will be passed to `msbuild`. For example, to build the Embree library in parallel use

```
cmake --build . --config Release --target embree -- /m
```

3.3.3 Building Embree - Using vcpkg

You can download and install Embree using the vcpkg dependency manager:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
./bootstrap-vcpkg.sh
./vcpkg integrate install
./vcpkg install embree3
```

The Embree port in vcpkg is kept up to date by Microsoft team members and community contributors. If the version is out of date, please create an issue or pull request on the vcpkg repository.

3.4 Windows SYCL Compilation

There are two options to compile Embree with SYCL support: The open source "oneAPI DPC++ Compiler" or the "Intel(R) oneAPI DPC++/C++ Compiler". Other SYCL compilers are not supported. You will also need an installed version of Visual Studio that supports the C++17 standard, e.g. Visual Studio 2019.

The "oneAPI DPC++ Compiler" is more up-to-date than the "Intel(R) oneAPI DPC++/C++ Compiler" but less stable. The current tested version of the oneAPI DPC++ compiler is

- oneAPI DPC++ Compiler 2023-10-26

Download and unpack the archive and open the "x64 Native Tools Command Prompt" of Visual Studio and execute the following lines to properly configure the environment to use the oneAPI DPC++ compiler:

```
set "DPCPP_DIR=path_to_dpcpp_compiler"
set "PATH=%DPCPP_DIR%\bin;%PATH%"
set "PATH=%DPCPP_DIR%\lib;%PATH%"
set "CPATH=%DPCPP_DIR%\include;%CPATH%"
set "INCLUDE=%DPCPP_DIR%\include;%INCLUDE%"
set "LIB=%DPCPP_DIR%\lib;%LIB%"
```

The `path_to_dpcpp_compiler` should point to the unpacked oneAPI DPC++ compiler.

Now, you can configure Embree using CMake by executing the following command in the Embree root directory:

```
cmake -B build
      -G Ninja
      -D CMAKE_BUILD_TYPE=Release
      -D CMAKE_CXX_COMPILER=clang++
      -D CMAKE_C_COMPILER=clang
      -D EMBREE_SYCL_SUPPORT=ON
      -D TBB_ROOT=path_to_tbb\lib\cmake\tbb
```

This will create a directory `build` to use as the CMake build directory, and configure a release build that uses `clang++` and `clang` from the oneAPI DPC++ compiler.

The Ninja generator is currently the easiest way to use the oneAPI DPC++ compiler.

We also enable SYCL support in Embree using the `EMBREE_SYCL_SUPPORT` CMake option.

Alternatively, you can download and run the installer of the

- Intel(R) oneAPI DPC++/C++ Compiler.

After installation, you can either open a regular `Command Prompt` and execute the `vars.bat` script in the `env` directory of the compiler install directory, for example

```
C:\Program Files (x86)\Intel\oneAPI\compiler\latest\env\vars.bat
```

or simply open the installed "Intel oneAPI command prompt for Intel 64 for Visual Studio".

Both ways will put the `icx` compiler executable from the Intel(R) oneAPI DPC++/C++ compiler in your path.

Now, you can configure Embree using CMake by executing the following command in the Embree root directory:

```
cmake -B build
      -G Ninja
      -D CMAKE_BUILD_TYPE=Release
      -D CMAKE_CXX_COMPILER=icx
      -D CMAKE_C_COMPILER=icx
      -D EMBREE_SYCL_SUPPORT=ON
      -D TBB_ROOT=path_to_tbb\lib\cmake\tbb
```

More information about setting up the Intel(R) oneAPI DPC++/C++ compiler can be found in the Development Reference Guide. Please note, that the Intel(R) oneAPI DPC++/C++ compiler requires at least CMake version 3.23 on Windows.

Independent of the DPC++ compiler choice, you can now build Embree using

```
cmake --build build
```

If you have problems with Ninja re-running CMake in an infinite loop, then first remove the "Re-run CMake if any of its inputs changed." section from the `build.ninja` file and run the above command again.

You can also create an Embree package using the following command:

```
cmake --build build --target package
```

Please see the Building Embree SYCL Applications section on how to build your application with such an Embree package.

3.4.1 Windows Graphics Driver Installation

In order to run the SYCL tutorials on HPG hardware, you first need to install the graphics drivers for your graphics card from `https://www.intel.com`. Please make sure to have installed version 31.0.101.4644 or newer.

3.5 CMake Configuration

The default CMake configuration in the configuration dialog should be appropriate for most usages. The following list describes all parameters that can be configured in CMake:

- `CMAKE_BUILD_TYPE`: Can be used to switch between Debug mode (Debug), Release mode (Release) (default), and Release mode with enabled assertions and debug symbols (RelWithDebInfo).
- `EMBREE_STACK_PROTECTOR`: Enables protection of return address from buffer overwrites. This option is OFF by default.
- `EMBREE_ISPC_SUPPORT`: Enables Intel® ISPC support of Embree. This option is OFF by default.
- `EMBREE_SYCL_SUPPORT`: Enables GPU support using SYCL. When this option is enabled you have to use some DPC++ compiler. Please see the sections Linux SYCL Compilation and Windows SYCL Compilation on supported DPC++ compilers. This option is OFF by default.
- `EMBREE_SYCL_AOT_DEVICES`: Selects a list of GPU devices for ahead-of-time (AOT) compilation of device code. Possible values are either, "none" which enables only just in time (JIT) compilation, or a list of the Embree-supported Xe GPUs for AOT compilation:
  - XE_HPG_CORE : Xe HPG devices
  - XE_HPC_CORE : Xe HPC devices

One can also specify multiple devices separated by comma to compile ahead of time for multiple devices, e.g. "XE_HPG_CORE,XE_HP_CORE". When enabling AOT compilation for one or multiple devices, JIT compilation will always additionally be enabled in case the code is executed on a device no code is precompiled for.

Execute "ocloc compile –help" for more details of possible devices to pass. Embree is only supported on Xe HPG/HPC and newer devices.

Per default, this option is set to "none" to enable JIT compilation. We recommend using JIT compilation as this enables the use of specialization constants to reduce code complexity.

- `EMBREE_STATIC_LIB`: Builds Embree as a static library (OFF by default). Further multiple static libraries are generated for the different ISAs selected (e.g. `embree4.a`, `embree4_sse42.a`, `embree4_avx.a`, `embree4_avx2.a`, `embree4_avx512.a`). You have to link these libraries in exactly this order of increasing ISA.
- `EMBREE_API_NAMESPACE`: Specifies a namespace name to put all Embree API symbols inside. By default, no namespace is used and plain C symbols are exported.
- `EMBREE_LIBRARY_NAME`: Specifies the name of the Embree library file created. By default, the name embree4 is used.
- `EMBREE_IGNORE_CMAKE_CXX_FLAGS`: When enabled, Embree ignores default CMAKE_CXX_FLAGS. This option is turned ON by default.
- `EMBREE_TUTORIALS`: Enables build of Embree tutorials (default ON).
- `EMBREE_BACKFACE_CULLING`: Enables backface culling, i.e. only surfaces facing a ray can be hit. This option is turned OFF by default.
- `EMBREE_BACKFACE_CULLING_CURVES`: Enables backface culling for curves, i.e. only surfaces facing a ray can be hit. This option is turned OFF by default.
- `EMBREE_BACKFACE_CULLING_SPHERES`: Enables backface culling for spheres, i.e. only surfaces facing a ray can be hit. This option is turned OFF by default.
- `EMBREE_COMPACT_POLYS`: Enables compact tris/quads, i.e. only geomIDs and primIDs are stored inside the leaf nodes.
- `EMBREE_FILTER_FUNCTION`: Enables the intersection filter function feature (ON by default).
- `EMBREE_RAY_MASK`: Enables the ray masking feature (OFF by default).
- `EMBREE_RAY_PACKETS`: Enables ray packet traversal kernels. This feature is turned ON by default. When turned on packet traversal is used internally and packets passed to rtcIntersect4/8/16 are kept intact in callbacks (when the ISA of appropriate width is enabled).
- `EMBREE_IGNORE_INVALID_RAYS`: Makes code robust against the risk of full-tree traversals caused by invalid rays (e.g. rays containing INF/NaN as origins). This option is turned OFF by default.
- `EMBREE_TASKING_SYSTEM`: Chooses between Intel® Threading TBB Building Blocks (TBB), Parallel Patterns Library (PPL) (Windows only), or an internal tasking system (INTERNAL). By default, TBB is used.

- EMBREE_TBB_ROOT: If Intel® Threading Building Blocks (TBB) is used as a tasking system, search the library in this directory tree.
- EMBREE_TBB_COMPONENT: The component/library name of Intel® Threading Building Blocks (TBB). Embree searches for this library name (default: tbb) when TBB is used as the tasking system.
- EMBREE_TBB_POSTFIX: If Intel® Threading Building Blocks (TBB) is used as a tasking system, link to tbb.(so,dll,lib). Defaults to the empty string.
- EMBREE_TBB_DEBUG_ROOT: If Intel® Threading Building Blocks (TBB) is used as a tasking system, search the library in this directory tree in Debug mode. Defaults to EMBREE_TBB_ROOT.
- EMBREE_TBB_DEBUG_POSTFIX: If Intel® Threading Building Blocks (TBB) is used as a tasking system, link to tbb.(so,dll,lib) in Debug mode. Defaults to "_debug".
- EMBREE_MAX_ISA: Select highest supported ISA (SSE2, SSE4.2, AVX, AVX2, AVX512, or NONE). When set to NONE the EMBREE_ISA_* variables can be used to enable ISAs individually. By default, the option is set to AVX2.
- EMBREE_ISA_SSE2: Enables SSE2 when EMBREE_MAX_ISA is set to NONE. By default, this option is turned OFF.
- EMBREE_ISA_SSE42: Enables SSE4.2 when EMBREE_MAX_ISA is set to NONE. By default, this option is turned OFF.
- EMBREE_ISA_AVX: Enables AVX when EMBREE_MAX_ISA is set to NONE. By default, this option is turned OFF.
- EMBREE_ISA_AVX2: Enables AVX2 when EMBREE_MAX_ISA is set to NONE. By default, this option is turned OFF.
- EMBREE_ISA_AVX512: Enables AVX-512 for Skylake when EMBREE_MAX_ISA is set to NONE. By default, this option is turned OFF.
- EMBREE_GEOMETRY_TRIANGLE: Enables support for triangle geometries (ON by default).
- EMBREE_GEOMETRY_QUAD: Enables support for quad geometries (ON by default).
- EMBREE_GEOMETRY_CURVE: Enables support for curve geometries (ON by default).
- EMBREE_GEOMETRY_SUBDIVISION: Enables support for subdivision geometries (ON by default).
- EMBREE_GEOMETRY_INSTANCE: Enables support for instances (ON by default).
- EMBREE_GEOMETRY_INSTANCE_ARRAY: Enables support for instance arrays (ON by default).
- EMBREE_GEOMETRY_USER: Enables support for user-defined geometries (ON by default).
- EMBREE_GEOMETRY_POINT: Enables support for point geometries (ON by default).

- `EMBREE_CURVE_SELF_INTERSECTION_AVOIDANCE_FACTOR`: Specifies a factor that controls the self-intersection avoidance feature for flat curves. Flat curve intersections which are closer than curve_radius*`EMBREE_CURVE_SELF_INTERSECTION_AVOIDANCE_FACTOR` to the ray origin are ignored. A value of 0.0f disables self-intersection avoidance while 2.0f is the default value.
- `EMBREE_DISC_POINT_SELF_INTERSECTION_AVOIDANCE`: Enables self-intersection avoidance for RTC_GEOMETRY_TYPE_DISC_POINT geometry type (ON by default). When enabled intersections are skipped if the ray origin lies inside the sphere defined by the point primitive.
- `EMBREE_MIN_WIDTH`: Enabled the min-width feature, which allows increasing the radius of curves and points to match some amount of pixels. See [rtcSetGeometryMaxRadiusScale] for more details.
- `EMBREE_MAX_INSTANCE_LEVEL_COUNT`: Specifies the maximum number of nested instance levels. Should be greater than 0; the default value is 1. Instances nested any deeper than this value will silently disappear in release mode, and cause assertions in debug mode.

Chapter 4
Embree Tutorials

Embree comes with a set of tutorials aimed at helping users understand how Embree can be used and extended. There is a very basic minimal that can be compiled as both C and C++, which should get new users started quickly. All other tutorials exist in an Intel® ISPC and C++ version to demonstrate the two versions of the API. Look for files named `tutorialname_device.ispc` for the Intel® ISPC implementation of the tutorial, and files named `tutorialname_device.cpp` for the single ray C++ version of the tutorial. To start the C++ version use the `tutorialname` executables, to start the Intel® ISPC version use the `tutorialname_ispc` executables. All tutorials can print available command line options using the `--help` command line parameter.

For all tutorials except minimal, you can select an initial camera using the `--vp` (camera position), `--vi` (camera look-at point), `--vu` (camera up vector), and `--fov` (vertical field of view) command line parameters:

```
./triangle_geometry --vp 10 10 10 --vi 0 0 0
```

You can select the initial window size using the `--size` command line parameter, or start the tutorials in full screen using the `--fullscreen` parameter:

```
./triangle_geometry --size 1024 1024
./triangle_geometry --fullscreen
```

The initialization string for the Embree device (`rtcNewDevice` call) can be passed to the ray tracing core through the `--rtcore` command line parameter, e.g.:

```
./triangle_geometry --rtcore verbose=2,threads=1
```

The navigation in the interactive display mode follows the camera orbit model, where the camera revolves around the current center of interest. With the left mouse button you can rotate around the center of interest (the point initially set with `--vi`). Holding Control pressed while clicking the left mouse button rotates the camera around its location. You can also use the arrow keys for navigation.

You can use the following keys:

F1 Default shading
F2 Gray EyeLight shading
F3 Traces occlusion rays only.
F4 UV Coordinate visualization
F5 Geometry normal visualization
F6 Geometry ID visualization
F7 Geometry ID and Primitive ID visualization

F8 Simple shading with 16 rays per pixel for benchmarking.
F9 Switches to render cost visualization. Pressing again reduces brightness.
F10 Switches to render cost visualization. Pressing again increases brightness.
f Enters or leaves full screen mode.
c Prints camera parameters.
ESC Exits the tutorial.
q Exits the tutorial.

4.1 Minimal

This tutorial is designed to get new users started with Embree. It can be compiled as both C and C++. It demonstrates how to initialize a device and scene, and how to intersect rays with the scene. There is no image output to keep the tutorial as simple as possible.

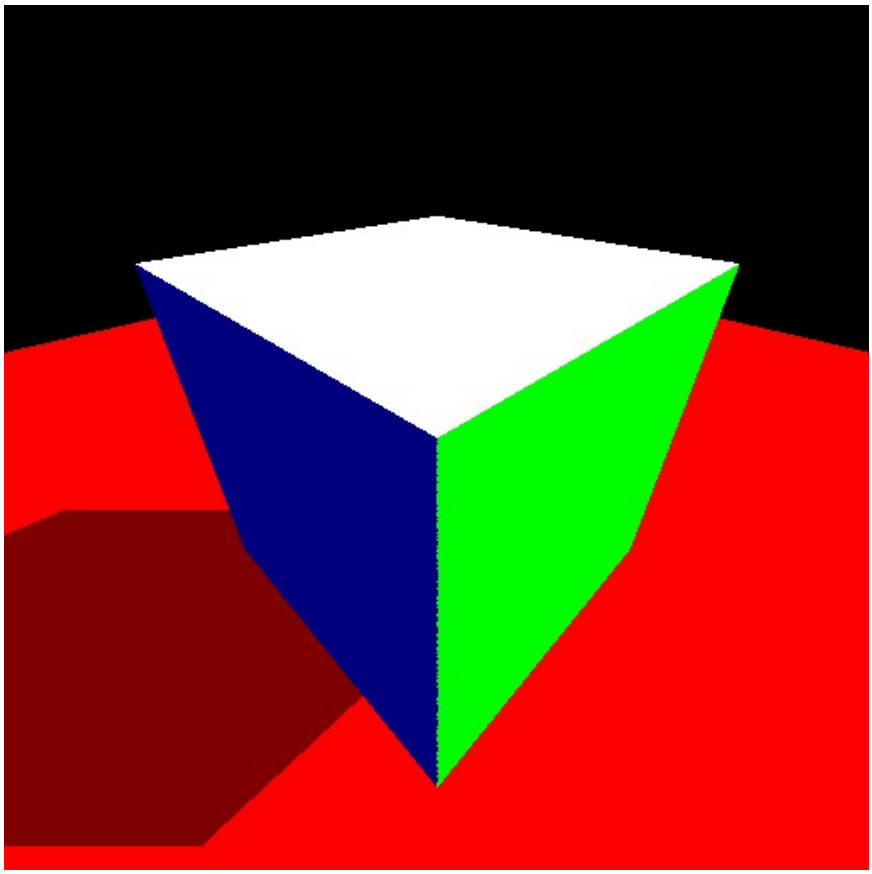
Source Code

4.2 Host Device Memory

This tutorial shows four different ways to use explicit host and device memory with SYCL.

Source Code

4.3 Triangle Geometry

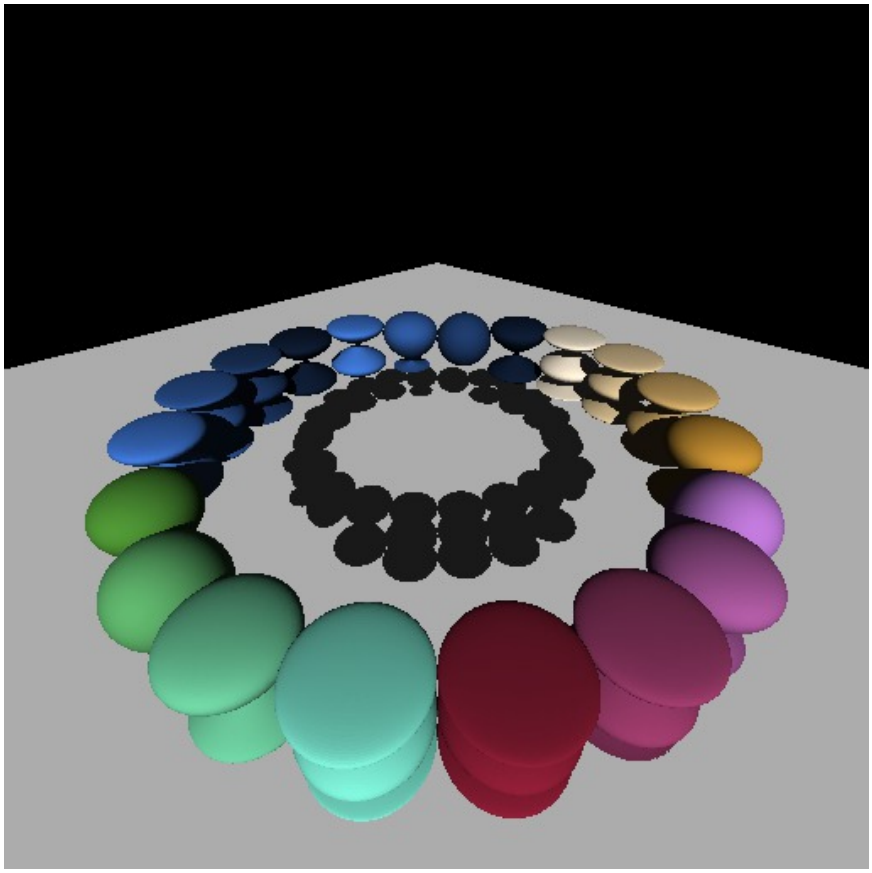


This tutorial demonstrates the creation of a static cube and ground plane using triangle meshes. It also demonstrates the use of the `rtcIntersect1` and

`rtcOccluded1` functions to render primary visibility and hard shadows. The cube sides are colored based on the ID of the hit primitive.

Source Code

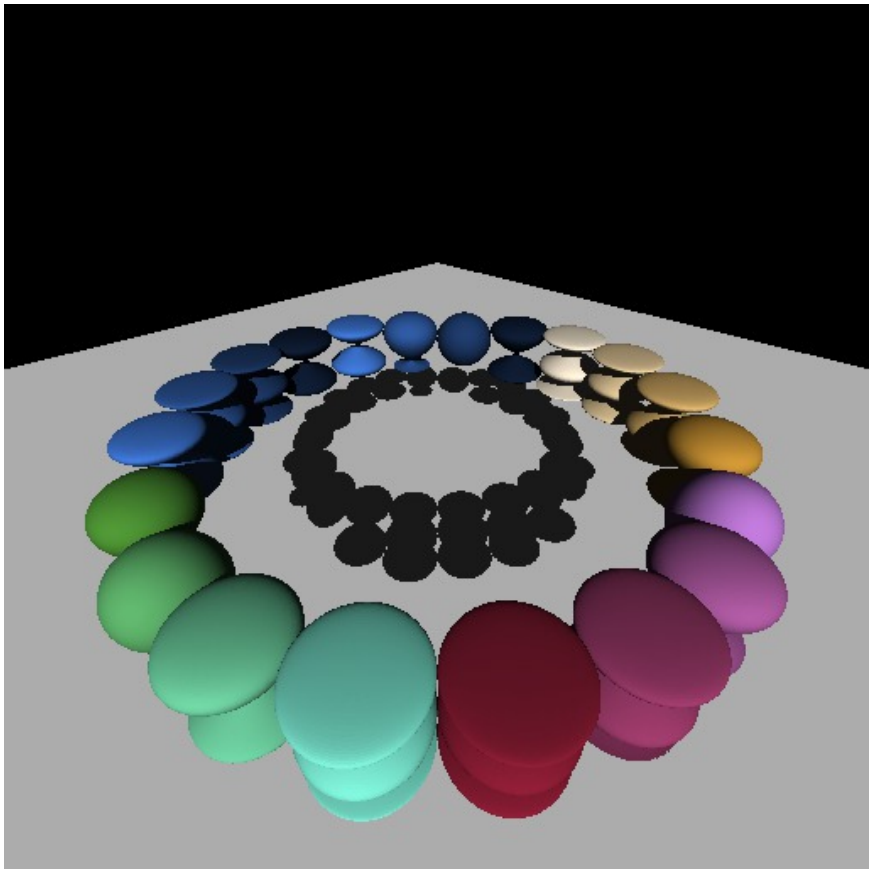
4.4 Dynamic Scene



This tutorial demonstrates the creation of a dynamic scene, consisting of several deforming spheres. Half of the spheres use the `RTC_BUILD_QUALITY_REFIT` geometry build quality, which allows Embree to use a refitting strategy for these spheres, the other half uses the `RTC_BUILD_QUALITY_LOW` geometry build quality, causing a high performance rebuild of their spatial data structure each frame. The spheres are colored based on the ID of the hit sphere geometry.

Source Code

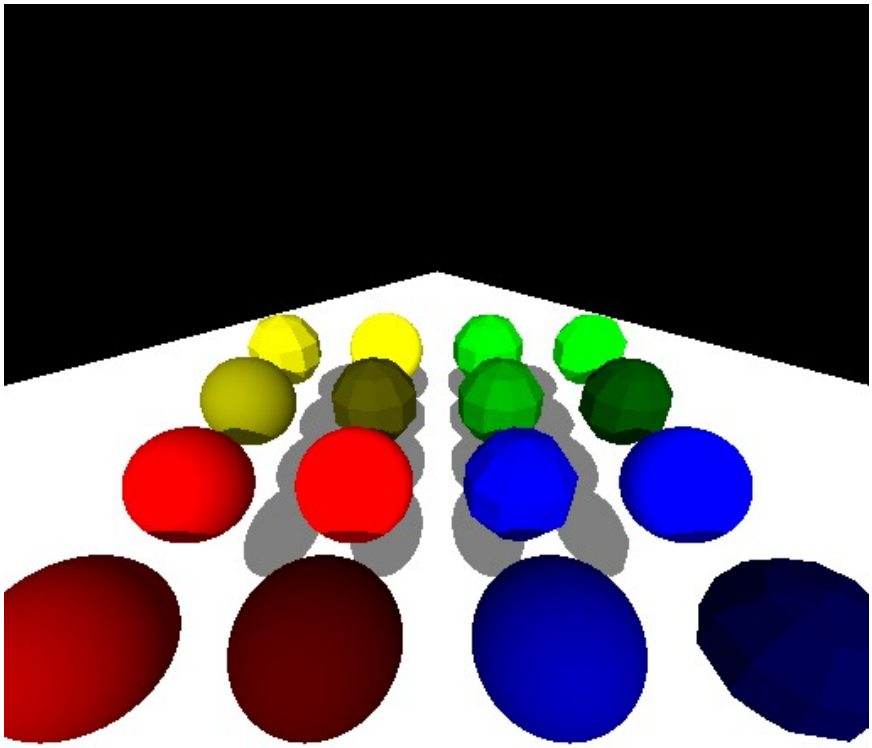
4.5 Multi Scene Geometry



This tutorial demonstrates the creation of multiple scenes sharing the same geometry objects. Here, three scenes are built. One with all the dynamic spheres of the Dynamic Scene test and two others each with half. The ground plane is shared by all three scenes. The space bar is used to cycle the scene chosen for rendering.

Source Code

4.6 User Geometry



This tutorial shows the use of user-defined geometry, to re-implement instancing, and to add analytic spheres. A two-level scene is created, with a triangle mesh as ground plane, and several user geometries that instance other scenes with a small number of spheres of different kinds. The spheres are colored using the instance ID and geometry ID of the hit sphere, to demonstrate how the same geometry instanced in different ways can be distinguished.

Source Code

4.7 Viewer

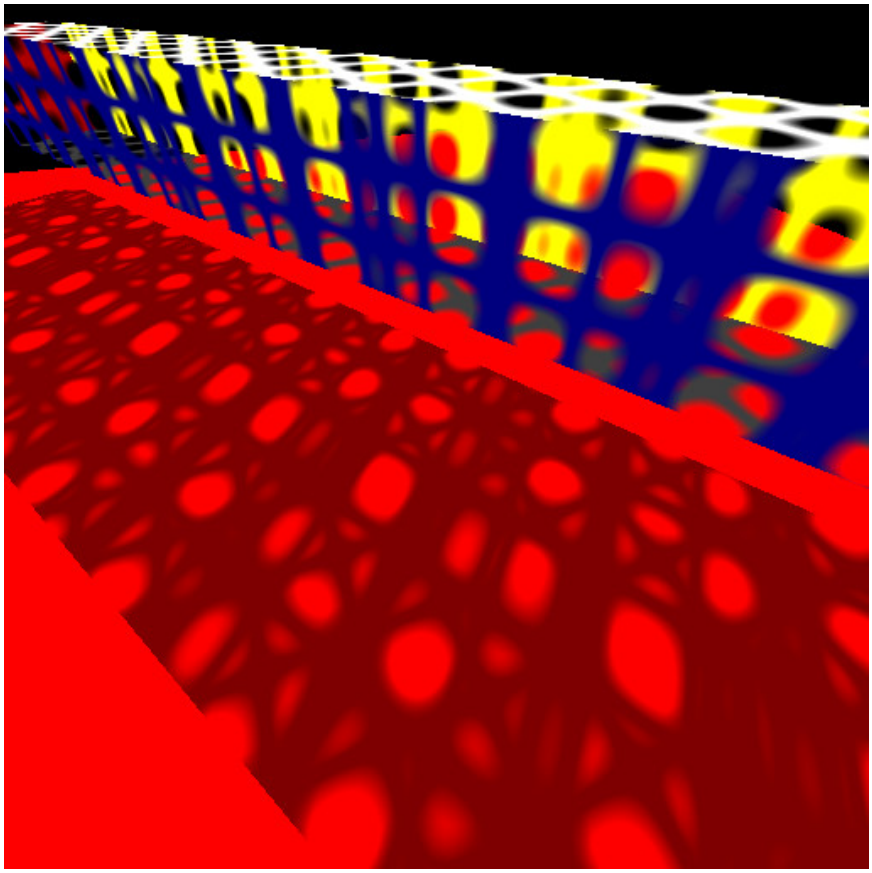
This tutorial demonstrates a simple OBJ viewer that traces primary visibility rays only. A scene consisting of multiple meshes is created, each mesh sharing the index and vertex buffer with the application. It also demonstrates how to support additional per-vertex data, such as shading normals.

You need to specify an OBJ file at the command line for this tutorial to work:

```
./viewer -i model.obj
```

Source Code

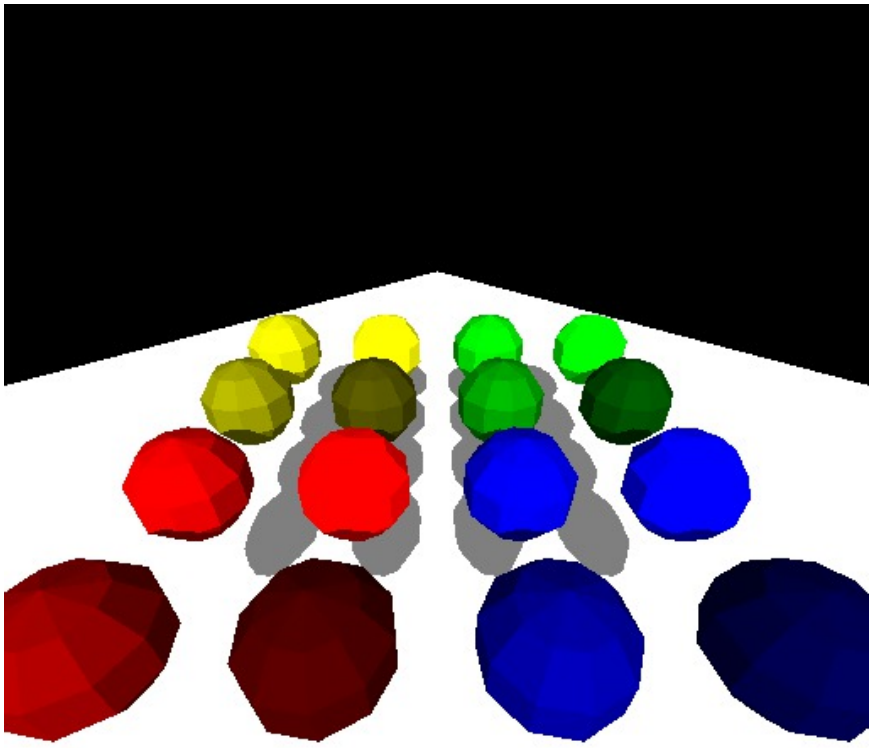
4.8 Intersection Filter



This tutorial demonstrates the use of filter callback functions to efficiently implement transparent objects. The filter function used for primary rays lets the ray pass through the geometry if it is entirely transparent. Otherwise, the shading loop handles the transparency properly, by potentially shooting secondary rays. The filter function used for shadow rays accumulates the transparency of all surfaces along the ray, and terminates traversal if an opaque occluder is hit.

Source Code

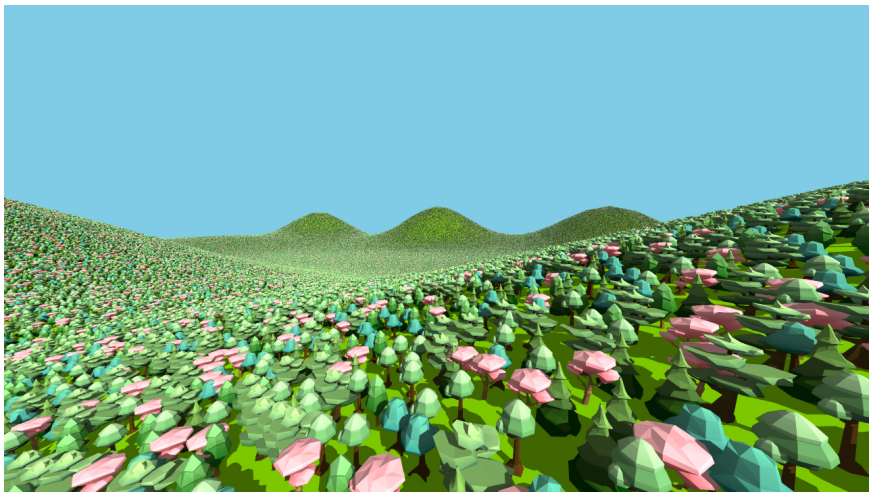
4.9 Instanced Geometry



This tutorial demonstrates the in-build instancing feature of Embree, by instancing a number of other scenes built from triangulated spheres. The spheres are again colored using the instance ID and geometry ID of the hit sphere, to demonstrate how the same geometry instanced in different ways can be distinguished.

Source Code

4.10 Instance Array Geometry



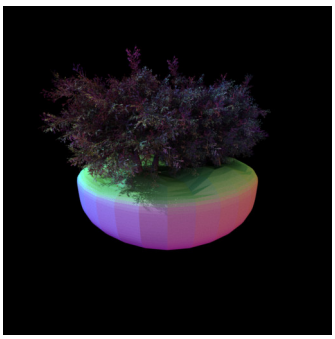
This tutorial demonstrates the usage of instance arrays in Embree. Instance

arrays are large collections of similar objects. Examples are sand dunes that consist of millions of instances of a few grain models or, like here, a forest consisting of many instances of a few tree models.

In this application can switch between representing the scene with regular instances or (one!) instance array. It also prints several stats, that demonstrate the memory savings and faster BVH build times when using instance arrays for such scenes. Instance arrays come with a small overhead on CPU and should be preferred if memory consumption is more important than raytracing performance.

Source Code

4.11 Multi Level Instancing



This tutorial demonstrates multi-level instancing, i.e., nesting instances into instances. To enable the tutorial, set the compile-time variable `EMBREE_MAX_INSTANCE_LEVEL_COUNT` to a value other than the default 1. This variable is available in the code as `RTC_MAX_INSTANCE_LEVEL_COUNT`.

The renderer uses a basic path tracing approach, and the image will progressively refine over time. There are two levels of instances in this scene: multiple instances of the same tree nest instances of a twig. Intersections on up to `RTC_MAX_INSTANCE_LEVEL_COUNT` nested levels of instances work out of the box. Users may obtain the *instance ID stack* for a given hitpoint from the `instID` member. During shading, the instance ID stack is used to accumulate normal transformation matrices for each hit. The tutorial visualizes transformed normals as colors.

Source Code

4.12 Path Tracer

This tutorial is a simple path tracer, based on the viewer tutorial.

You need to specify an OBJ file and light source at the command line for this tutorial to work:

```
./pathtracer -i model.obj --ambientlight 1 1 1
```

As example models we provide the "Austrian Imperial Crown" model by Martin Lubich and the "Asian Dragon" model from the Stanford 3D Scanning Repository.

crown.zip
asian_dragon.zip

To render these models execute the following:

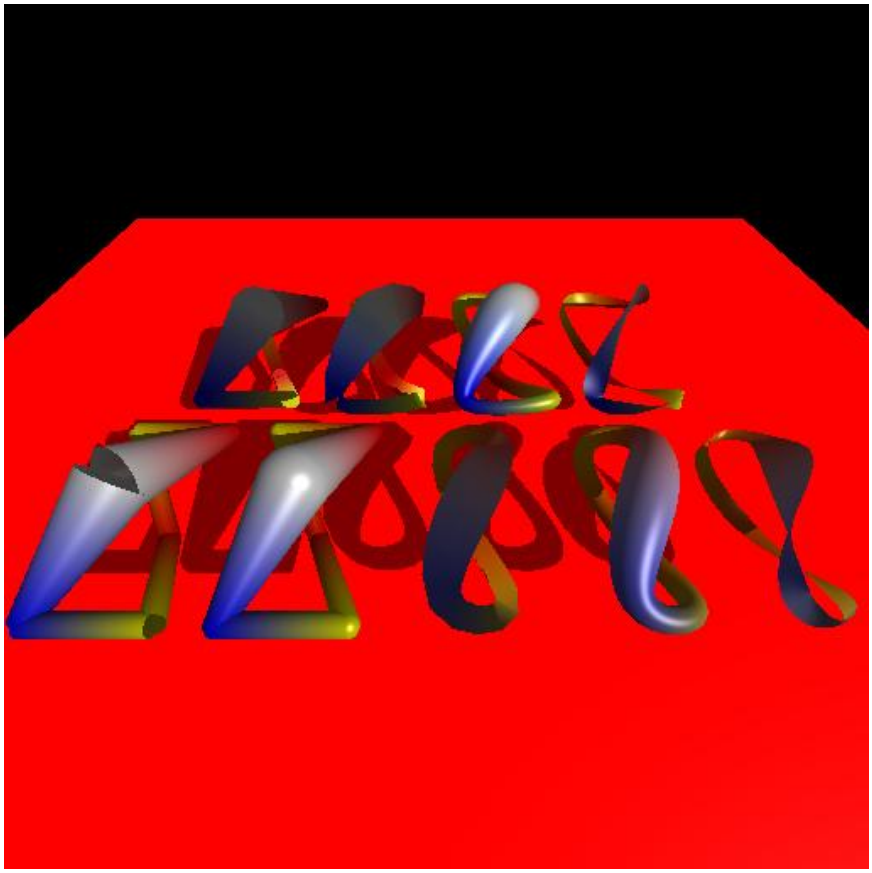
```
./pathtracer -c crown/crown.ecs
./pathtracer -c asian_dragon/asian_dragon.ecs
```

Source Code

4.13 Hair

This tutorial demonstrates the use of the hair geometry to render a hairball.
Source Code

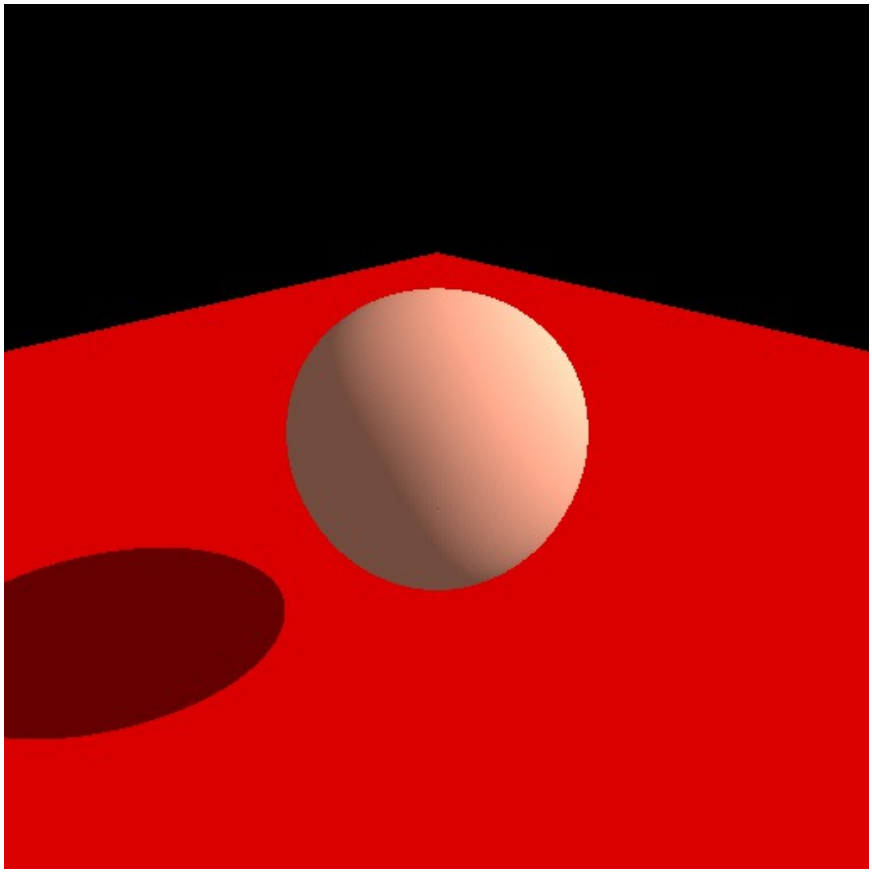
4.14 Curve Geometry



This tutorial demonstrates the use of the Linear Basis, B-Spline, and Catmull-Rom curve geometries.

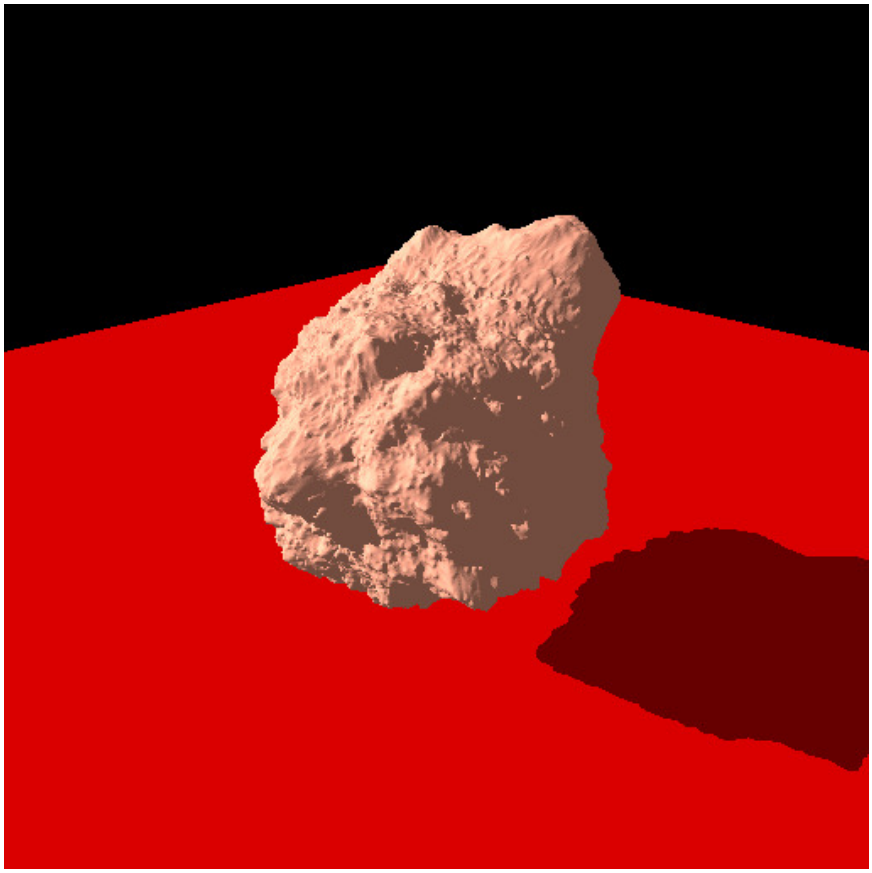
Source Code

4.15 Subdivision Geometry



This tutorial demonstrates the use of Catmull-Clark subdivision surfaces.
Source Code

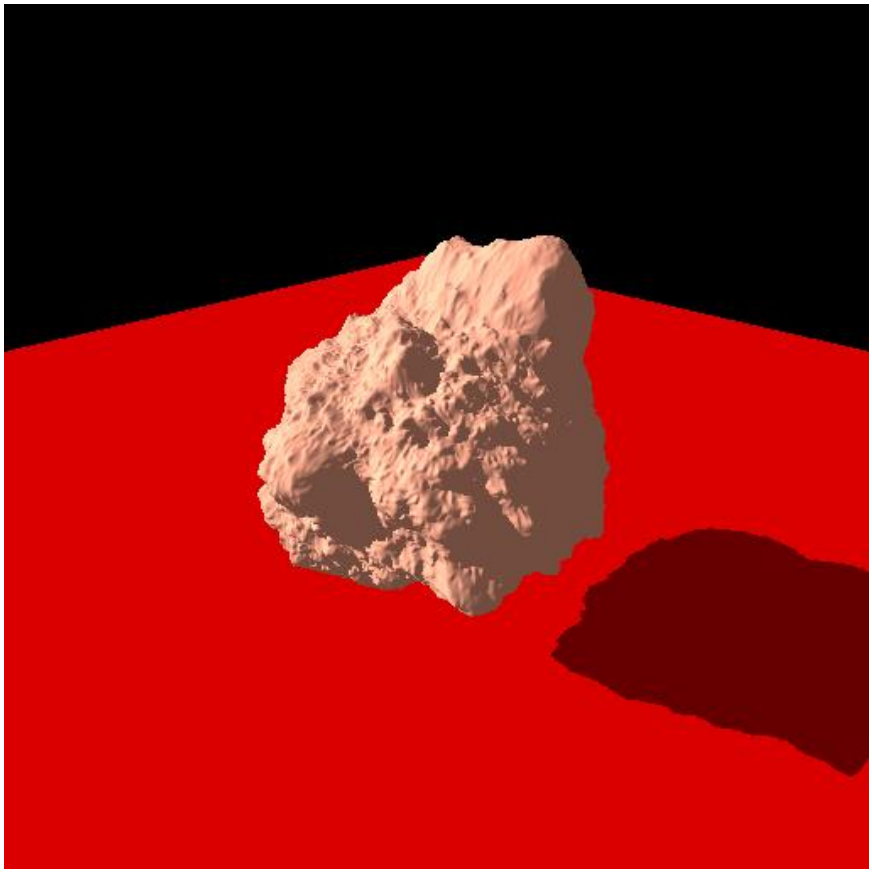
4.16 Displacement Geometry



This tutorial demonstrates the use of Catmull-Clark subdivision surfaces with procedural displacement mapping using a constant edge tessellation level.

Source Code

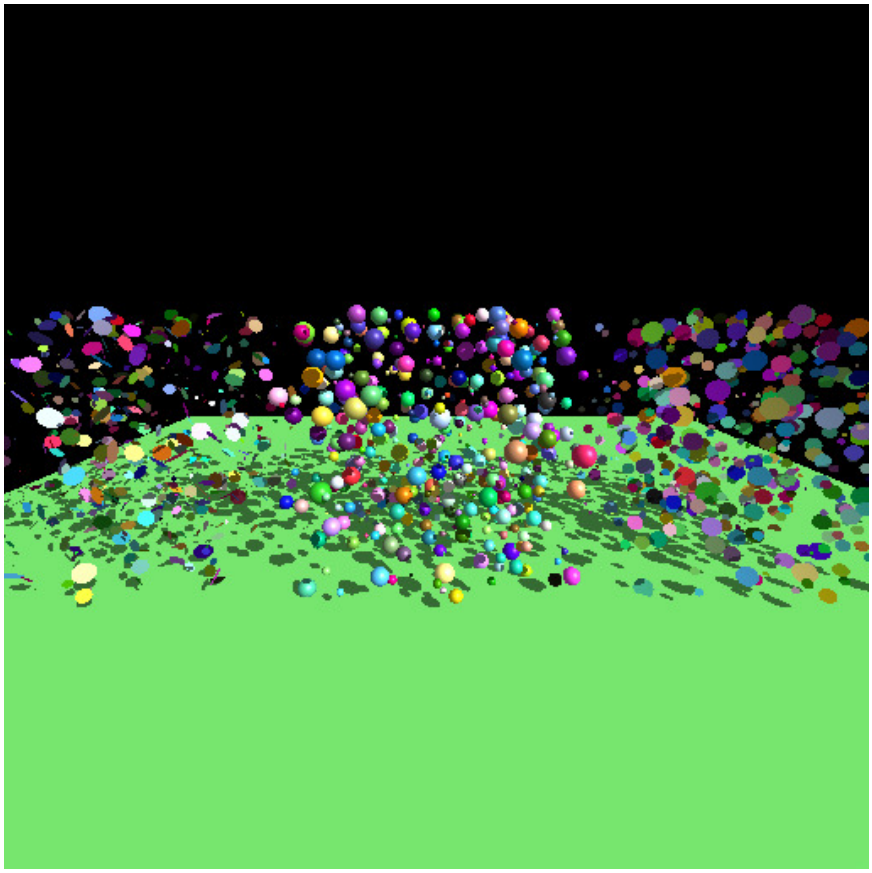
4.17 Grid Geometry



This tutorial demonstrates the use of the memory efficient grid primitive to handle highly tessellated and displaced geometry.

Source Code

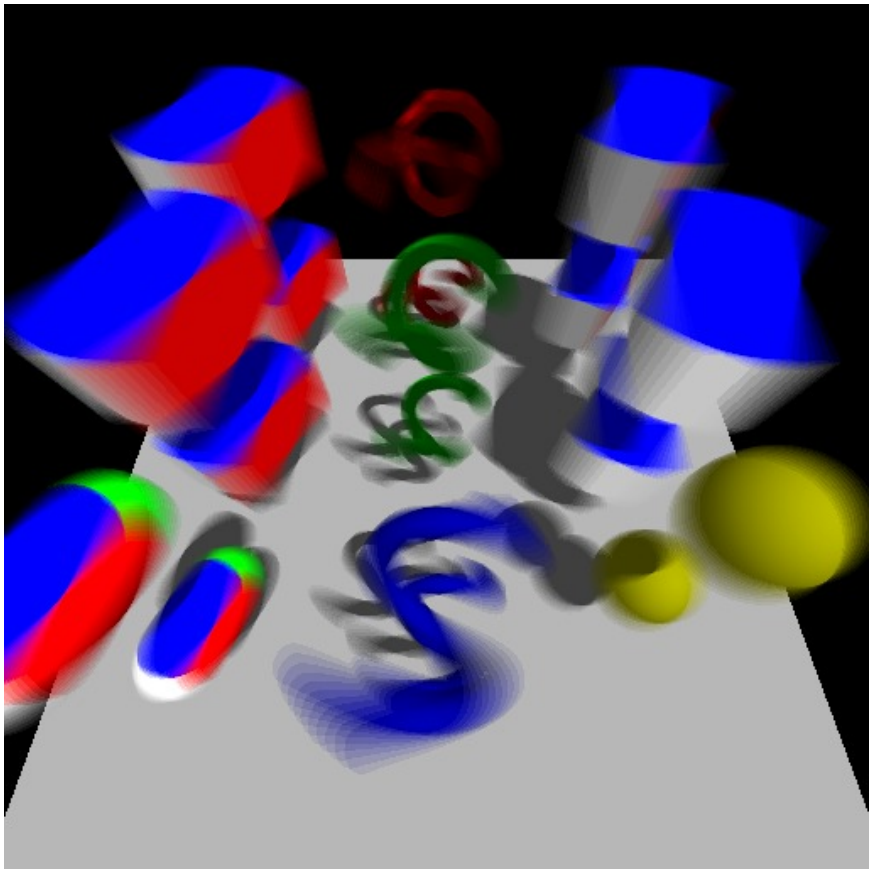
4.18 Point Geometry



This tutorial demonstrates the use of the three representations of point geometry.

Source Code

4.19 Motion Blur Geometry

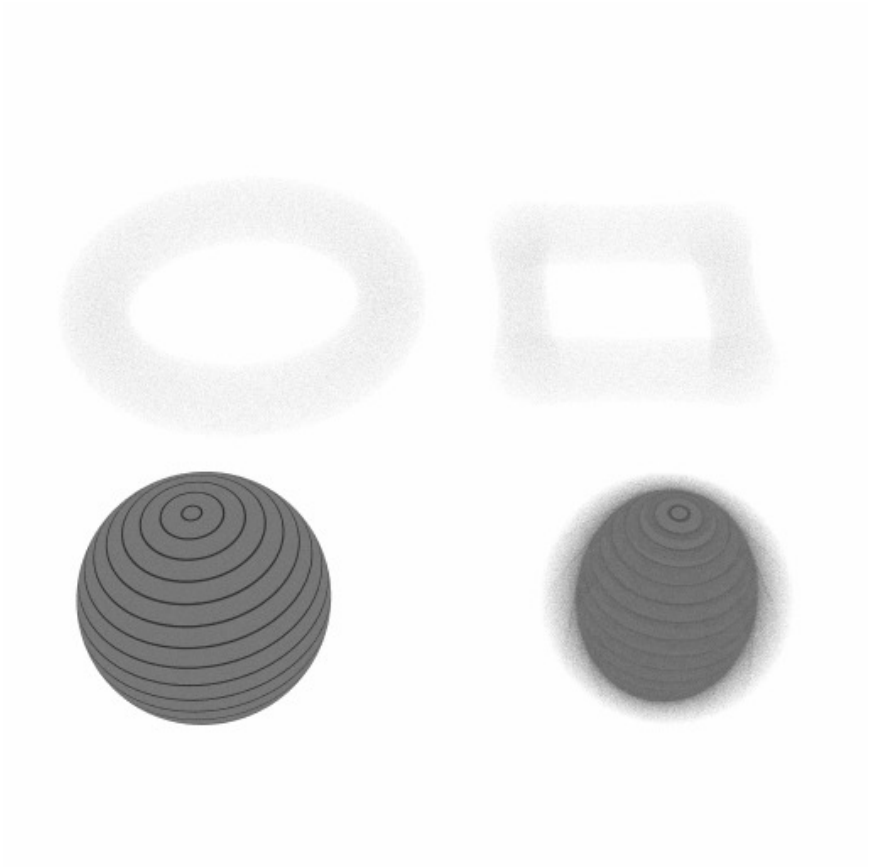


This tutorial demonstrates rendering of motion blur using the multi-segment motion blur feature. Shown is motion blur of a triangle mesh, quad mesh, subdivision surface, line segments, hair geometry, Bézier curves, instantiated triangle mesh where the instance moves, instantiated quad mesh where the instance and the quads move, and user geometry.

The number of time steps used can be configured using the `--time-steps <int>` and `--time-steps2 <int>` command line parameters, and the geometry can be rendered at a specific time using the the `--time <float>` command line parameter.

Source Code

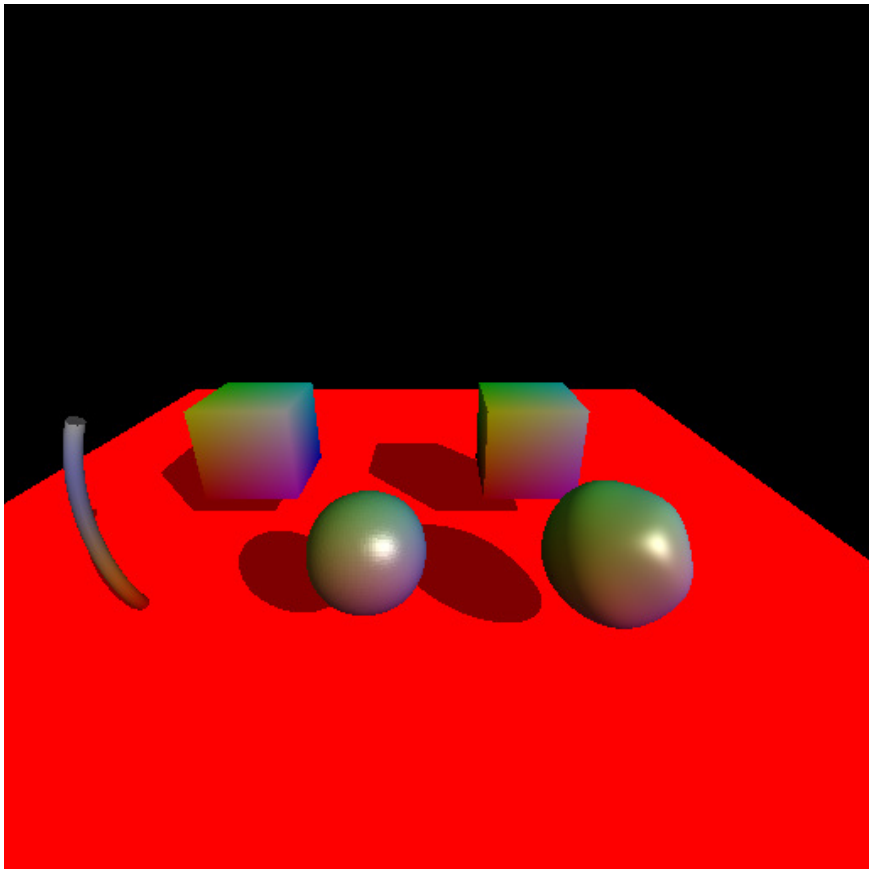
4.20 Quaternion Motion Blur



This tutorial demonstrates rendering of motion blur using quaternion interpolation. Shown is motion blur using spherical linear interpolation of the rotational component of the instance transformation on the left and simple linear interpolation of the instance transformation on the right. The number of time steps can be modified as well.

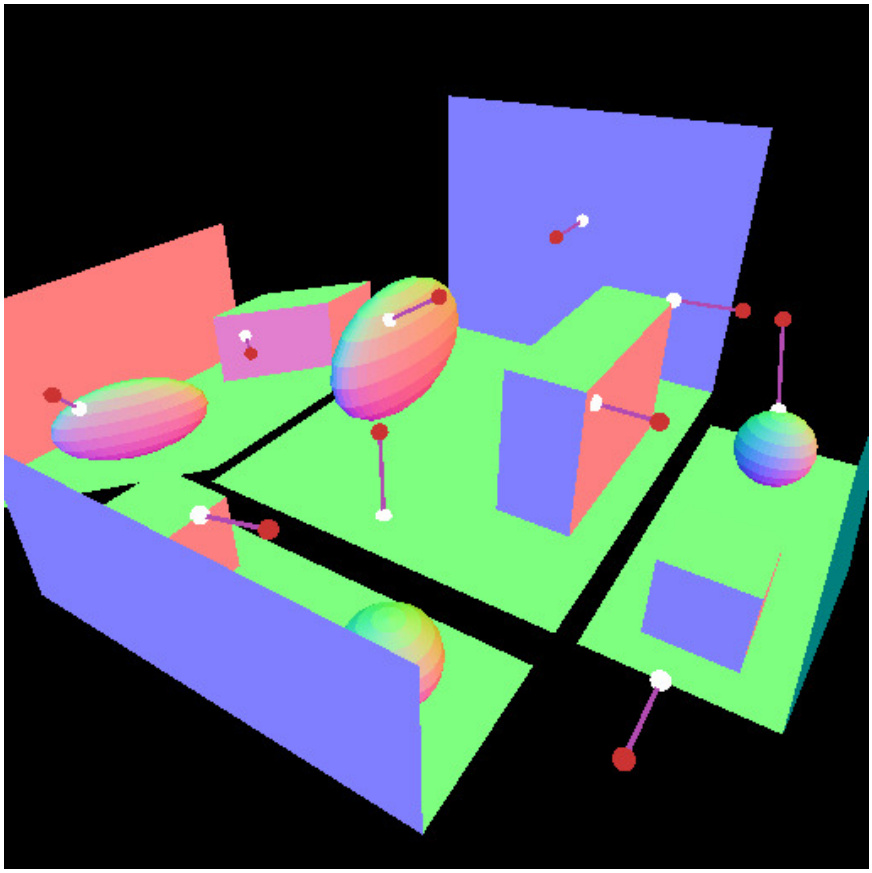
Source Code

4.21 Interpolation



This tutorial demonstrates interpolation of user-defined per-vertex data.
Source Code

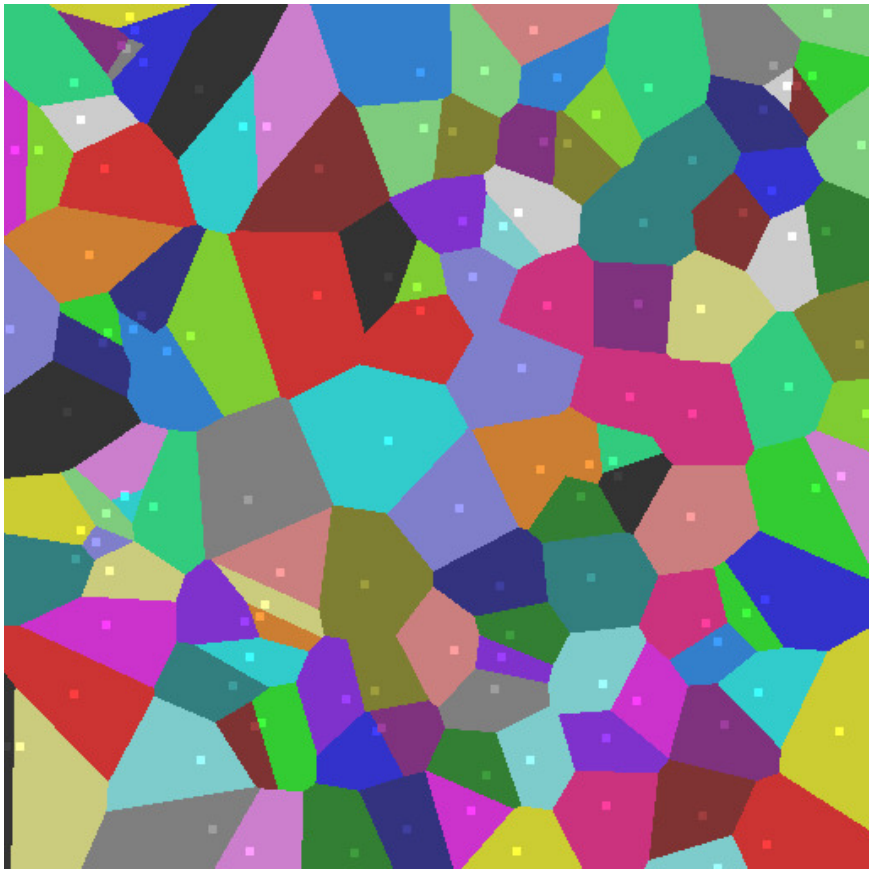
4.22 Closest Point



This tutorial demonstrates a use-case of the point query API. The scene consists of a simple collection of objects that are instanced and for several point in the scene (red points) the closest point on the surfaces of the scene are computed (white points). The closest point functionality is implemented for Embree internal and for user-defined instancing. The tutorial also illustrates how to handle instance transformations that are not similarity transforms.

Source Code

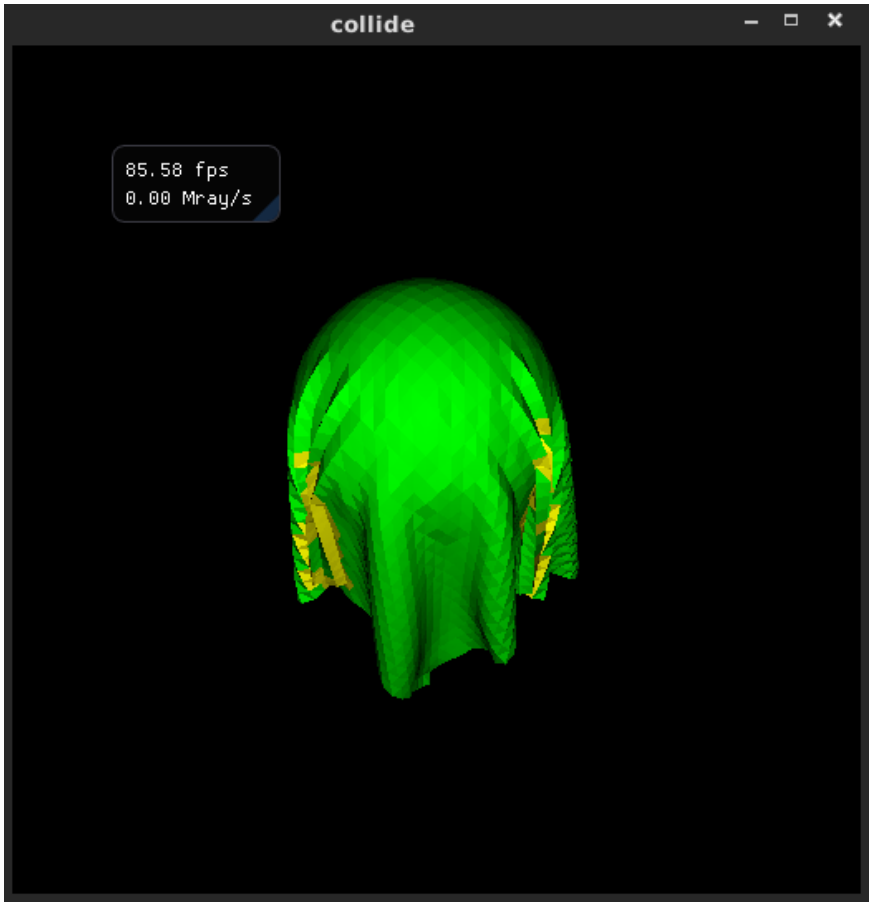
4.23 Voronoi



This tutorial demonstrates how to implement nearest neighbour lookups using the point query API. Several colored points are located on a plane and the corresponding voroni regions are illustrated.

Source Code

4.24 Collision Detection



This tutorial demonstrates how to implement collision detection using the collide API. A simple cloth solver is setup to collide with a sphere.

The cloth can be reset with the `space` bar. The sim stepped once with `n` and continuous simulation started and paused with `p`.

Source Code

4.25 BVH Builder

This tutorial demonstrates how to use the templated hierarchy builders of Embree to build a bounding volume hierarchy with a user-defined memory layout using a high-quality SAH builder using spatial splits, a standard SAH builder, and a very fast Morton builder.

Source Code

4.26 BVH Access

This tutorial demonstrates how to access the internal triangle acceleration structure build by Embree. Please be aware that the internal Embree data structures might change between Embree updates.

Source Code

4.27 Find Embree

This tutorial demonstrates how to use the `FIND_PACKAGE` CMake feature to use an installed Embree. Under Linux and macOS the tutorial finds the Embree installation automatically, under Windows the `embree_DIR` CMake variable must be set to the following folder of the Embree installation: `C:\Program Files\Intel\Embree3`.

Source Code

4.28 Next Hit

This tutorial demonstrates how to robustly enumerate all hits along the ray using multiple ray queries and an intersection filter function. To improve performance, the tutorial also supports collecting the next N hits in a single ray query.

Source Code