NVIDIA

NVIDIA Iray

Light path expressions

3 October 2024
Version 1.0

NVIDIA Iray – Light path expressions

 Document build number rev377400.3959

Contents

1 Light path expressions

1.1 Introduction

Light path expressions (LPEs) describe the propagation of light through a scene, for example starting from a source of light, bouncing around between the objects of the scene and ultimately ending up at the eye. The paths that light takes through a scene are called *light transport paths*. LPEs may be used for example, to extract only specific light contributions from a renderer into separate image buffers.

Light path expressions were first proposed as a description of light transport by Paul Heckbert [Heckbert90] (page 8). Heckbert suggested that *regular expressions* — typically used to describe patterns of characters in text — could also describe light transport events and paths. The alphabet of LPEs consists of event descriptions, that is, of interactions between light particles and the scene.

1.2 Events

Each event of a path, that is, each interaction of light with the scene objects and materials, is described by its type (for example, emission or reflection), the mode of scattering (for example, diffuse or specular), and an optional handle.

A full event is described as `< t m h >`, where

- `t` is the event type, either `R` (reflection), `T` (transmission), or `V` (volume interaction),
- `m` is the scattering mode, either `D` (diffuse), `G` (glossy), or `S` (specular), and
- `h` is a handle in single quotes, for example. `'foo'`. This position may be omitted from the specification. In that case, any handle is accepted. See below for details.

Spaces are ignored unless they occur inside a handle string. The dot character (`.`) may be used as a wildcard in any position. It accepts any valid input. For example, a diffuse reflection event may be specified as `<RD.>`, or, omitting the handle, `<RD>`. A specular transmission event identified with the handle `"window"` may be specified as `<TS'window'>`.

1.3 Handles

Handles are strings of ASCII characters, enclosed in single quotes (`'`). The following characters must be escaped by prefixing them with a backslash inside handles: `\`, `'`, and `"`. The assignment of a handle as a name for a scene element is typically made possible through the graphical interface of an application.

1.4 Sets and exclusion

As an alternative to the type, mode, and handle specifiers described above, each position of the event triple may contain a character set enclosed in square brackets. Any element of the

set will be accepted. For example, `<[RT]..>` matches all reflection events and all transmission events.

The complementary set is specified by first including the caret (^) character. For example, `<.[^S]>` matches any non-specular event and `<..[^'ground']>` matches any event that is not identified with the handle "ground".

Event sets also work on full events. For instance, `[<RG><TS>]` matches glossy reflection and specular transmission events. Note that this is different from `<[RT][GS]>`, which accepts glossy transmission and specular reflection in addition to the events accepted by the previous expression.

1.5 Abbreviations

In order to make specification of LPEs simpler, event descriptions may be abbreviated. An event in which only one of type, mode, or handle is explicitly specified may be replaced by that item. For example, `<R..>` may be abbreviated as `R`. Likewise, `<..'foo'>` may be abbreviated as `'foo'`. Note the difference between `<TS.>`, which accepts a single specular transmission event, and `TS`, which accepts an arbitrary transmission event followed by an arbitrary specular event. Finally, `.` matches any event except the special events described below.

Abbreviation rules also apply to event sets, that is, `[<T..><.S.>]` reduces to `[TS]`. Again note that this is different from `TS` (without brackets).

1.6 Constructing expressions

LPEs may be constructed by combining event specifications through concatenation, alternation, and quantification. The following operators are supported and have the same semantics as they would for standard regular expressions. For expressions `A` and `B` and integers `n` and `m` with `m >= n`:

`AB`	Accepts first `A`, then `B`
`A|B`	Accepts `A` or `B`
`A?`	Optionally accepts `A`, that is, `A` may or may not be present
`A*`	Accepts any number of occurrences of `A` in sequence, including zero times
`A+`	Accepts any non-empty sequence of `A`'s. It is equivalent to `AA*`
`A{n}`	Accepts exactly `n` consecutive occurrences of `A`. For example, `A{3}` is equivalent to `AAA`.
`A{n,m}`	Accepts from `n` to `m`, inclusively, occurrences of `A`
`A{n,}`	Equivalent to `A{n}A*`

The precedence from high to low is quantifiers (`?`, `*`, `+`, `{}`), concatenation, alternatives. Items can be grouped using normal parentheses `(` and `)`.

1.7 Special events

Each LPE is delimited by special events for the eye (or camera) and light vertices. These events serve as markers and must be the first and last symbols in a LPE.

LPEs may be specified starting either at the light or at the eye. All expressions must be constructed in such a way that every possible match has exactly one eye and one light vertex. This is due to the nature of LPEs: They describe light transport paths between one light and the eye. Note that this does not mean that light and eye markers must each show up exactly once. For example, `"E (D La | G Le)"` is correct, because either alternative has exactly one light and one eye marker: `"E D La"` and `"E G Le"`. On the other hand `"E D La?"`, `"E (D | La)"`, and `"E (D | La) Le"` are ill-formed, because they would match paths with zero or two light markers.

In the abbreviated form, the eye marker is simply `E` and the light marker is `L`. These items are special in that they represent two distinct characteristics: the shape of the light (or camera lens) and the mode of the emission distribution function. The full notation therefore differs from that of the standard events.

In the full form, a light source as the first vertex of a transport path is described as `< L h m h >`, where `L` is the light type, and `m` and `h` are as before. The first pair of type and handle describes the light source itself. The type `L` can be one of `Lp` (point shape), `La` (area light), `Le` (environment or background), `Lm` (matte lookup), `Lv` (emissive volume) or `L` (any type). In the case of alpha expressions, `Lms` (matte shadows) is supported in addition to the aforementioned types. The second pair describes the light's directional characteristics, that is, its EDF. (This form loosely corresponds to the full-path notation introduced by Eric Veach in [Veach97] (page 8), Section 8.3.2.)

As before, the handles are optional. Furthermore, the EDF specification may be omitted. Thus, `<La>` is equivalent to `<La...>` and `La`.

Especially when dealing with irradiance (rather than radiance) render targets, it is convenient to use a special form of LPE, called *irradiance expression*. Such expressions contain an irradiance marker, rather than an eye marker. Using this marker, it is possible to describe light transport up to the point of the irradiance measurement, rather than up to the camera.

The full form of the irradiance marker is `<I h>`, the abbreviated form is simply `I`. As before, `h` represents an optional handle. If set, irradiance will only be computed on those surfaces that have a matching handle.

1.8 Advanced operations

Several operations exist in order to make specifying the right expression easier. These operations do not add expressive power, but simplify certain tasks.

When an application provides a means for multiple output images (or *canvases*) to be rendered at the same time, expressions may be re-used in subsequent canvas names. This is achieved by assigning a name to the expressions that should be re-used, for example:

1. `caustics: L.*SDE`
2. `LE | $caustics`

In this example, the second canvas will receive both caustics and directly visible light sources. As illustrated above, variables are introduced by specifying the desired name, followed by a colon and the desired expression. Variable names may contain any positive number of alphanumeric characters and the underscore character, with the limitation that they may not start with a terminal symbol. Since all terminals of the LPE grammar start with capital letters, it is good practice to start variable names with lowercase letters. Note that sub-expressions cannot be captured by variable names.

Variables are referenced by applying the dollar (or value-of) operator to the variable name.

Expressions may be turned into their complement by prefixing them with the caret symbol. An expression of type `^A` will yield all light transport paths that are not matched by `A`. Note that the complement operator cannot be applied to sub-expressions: `"^(L.*E)"` is valid, but `"L^(.*)E"` is not.

It is possible to compute the intersection of two or more expressions by combining them with the ampersand symbol. Expressions of type `A & B` will match only those paths which are matched by both `A` and `B`.

1.9 Matte object interaction

The color of matte objects is determined by two types of interaction. The first is the lookup into the environment or backplate. This contribution is potentially shadowed by other objects in the scene and may be selected by expressions ending in `Lm`. Selection of such contributions can be further refined by specifying the handle of the matte object, for example, `<Lm 'crate'>`.

The second type of contributions is made up of effects that synthetic objects and lights have on matte objects. This includes effects like synthetic lights illuminating matte objects and synthetic objects reflected by matte objects. For these contributions, matte objects behave exactly like synthetic objects.

With regards to LPEs, matte lights illuminating synthetic objects behave exactly as if they were synthetic lights.

1.10 LPEs for alpha control

Some additional considerations are necessary when using LPEs to control alpha output.

By definition, alpha is transparent (alpha 0) only for paths that match the provided expression. Note that this means that light transport paths which do not reach a light source or the environment because they are terminated prematurely (for whatever reason) are opaque. This is necessary to avoid undesired semi-transparent areas in the standard case.

This has implications for the creation of object masks. Since the mask is supposed to be completely transparent also when undesired objects are hit by camera rays, these paths have to be captured by the expression even if they are terminated. This requires a special type of LPE, that is, one that captures terminated paths. Such LPEs are only allowed for alpha channels. For example, the expression `"E ([^'crate'] .*)? L?"` will render a mask for the object `'crate'`.

Shadows received by matte objects may also affect the opacity of the alpha channel. This opacity can be removed by capturing paths which end in `Lms`.[1] Adding opacity in areas of matte shadow to the previously shown mask expression may be achieved by slightly changing the LPE to `"E ([^'crate'] .*) L? | E [^Lms]"`.

Note that presence or absence of `Lms` controls whether shadows which are received by a certain matte object make the alpha channel opaque. This affects all matte shadow, regardless of how it was cast, and by which objects.

The API provides functions mi::neuraylib::IRendering_configuration::make_alpha_expression() and mi::neuraylib::IRendering_configuration::make_alpha_mask_expression() to generate various common alpha expressions, including masks.

1.11 Example LPEs

The universal light path expression, `"L .* E"`, accepts all light transport paths. By default, this expression yields the same result as not using LPEs at all. Remember that this is equivalent to `"L.*E"` (whitespace is ignored) and `"E.*L"` (the expression can be reversed).

Direct illumination can be requested by specifying `"L .? E"`, or `"L . E"` if directly visible light sources are not desired. Indirect illumination is then specified by `"L .{2,} E"`.

Compositing workflows often use the concept of diffuse and reflection passes. They can be specified with the LPEs `"E <RD> L"` and `"E <RS> L"`, respectively. Note that these passes as specified above do not contain indirect illumination. `"E <RD> .* L"` extends this to global illumination where the visible surfaces are diffuse. If only diffuse interactions are desired, `"E <RD>* L"` can be used.

Caustics are usually described as `"E D S .* L"`. The expression `"E D (S|G) .* L"` or `"E D [GS] .* L"` also considers glossy objects as caustics casters. If, for example, only specular reflection caustics cast by an object identified with a handle "crate" are desired, the expression is changed to `"E D <RS'crate'> .* L"`. This can further be restricted to caustics cast onto "ground" from a spot light by changing the expression to `"E 'ground' <RS'crate'> .* <LpG>"`.

Assuming an expression variable called `caustics` was defined in a previous expression, `"L.{2,5}E & ^$caustics"` will match any path that has the specified length and is not a caustic.

1.12 Summary of typical LPEs

Typical production workflow structures in digital compositing often employ a set of standard elements that can be represented by light path expressions. The following expressions define the color (RGB) component of rendering:

1. The mask expression in the previous paragraph captures all light types and thus also makes matte shadows transparent.

LPE	*Description*
`E D .* L`	Diffuse pass commonly used in conventional compositing workflows. The last event on the light path before the eye was a diffuse event.
`E G .* L`	Glossy pass commonly used in conventional compositing workflows. The last event on the light path before the eye was a glossy event.
`E S .* L`	Specular pass commonly used in conventional compositing workflows. The last event on the light path before the eye was a specular event, that is, a mirror reflection or specular refraction.
`E D S .* L`	Diffuse part of caustics cast by a mirror reflection or specular refraction on another surface.
`E .* <L'key'>`	All direct and indirect light contribution coming from the `key` light group, which are all lights in the scene with the `handle` attribute set to `key`.
`E 'crate' .* L`	All direct and indirect light falling onto any object in the scene with the `handle` attribute set to `crate`.

The alpha channel can also be specified by light path expressions:

LPE	*Description*
`E [LmLe]`	Alpha is based solely on primary visibility. This is the approach used traditionally by many renderers.
`E T* [LmLe]`	Transmitting objects make the alpha channel transparent. This is the default behavior.
`E <TS>* [LmLe]`	Only specular transmission makes the alpha channel transparent. This avoids unexpected results in scenes with materials that have a diffuse transmission component.

The graphical interface of an application may provide a way of naming and storing LPEs for reuse. Common LPEs like the above may also be part of a standard set of named LPEs in an application interface.

1.13 Light path expression grammar

`L`	light
`E`	eye
`R`	reflection type
`T`	transmission type
`V`	volume interaction type
`D`	diffuse mode
`G`	glossy mode
`S`	specular mode
'h'	handle *h*
`<` *type mode handle* `>`	event
`Lp`	point light type
`La`	area light type
`Lv`	emissive volume
`Le`	environment or background light type
`Lm`	matte lookup type
`Lms`	shadows cast onto matte objects (alpha expressions only)
`<` *light-type light-handle mode handle* `>`	light source full form
`<I` *h* `>`	irradiance marker
type	abbreviation for `<` *type* `..>`
mode	abbreviation for `<.` *mode* `.>`
handle	abbreviation for `<..` *handle* `>`
`I`	abbreviation for `<I.>`
`.`	match anything (in context)
`[` *A* ... `]`	match any element in set
`[^` *A* `]`	match all but *A*
AB	*A* followed by *B*
A`|`*B*	*A* or *B*
A`?`	zero or one *A*
A`*`	zero or more *A*s
A`+`	one or more *A*s
A`{`*n*`}`	a sequence of *n* *A*s
A`{`*n*`,`*m*`}`	*n* to *m* occurences of *A*
A`{`*n*`,}`	equivalent to *A*`{`*n*`}`*A*`*`
`(` ... `)`	grouping
`^` *expression*	complement of *expression*
expression-1 `&` *expression-2*	match both expressions
name`:` *expression*	assign *expression* to *name*
`$`*name*	use value of *name*

2 Bibliography

Paul Heckbert. “Adaptive Radiosity Textures for Bidirectional Ray Tracing.” *Computer Graphics* (SIGGRAPH ’90 Proceedings), vol. 24, no. 4, August 1990, pp. 145–154.

Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD Thesis, Stanford University, 1997.