Overview

- Phase 1: Implement LPE Engine
- Phase 2: LPE & Mitsuba interface (with Dr.Jit)
- Phase 3: Integrator modification & Validation

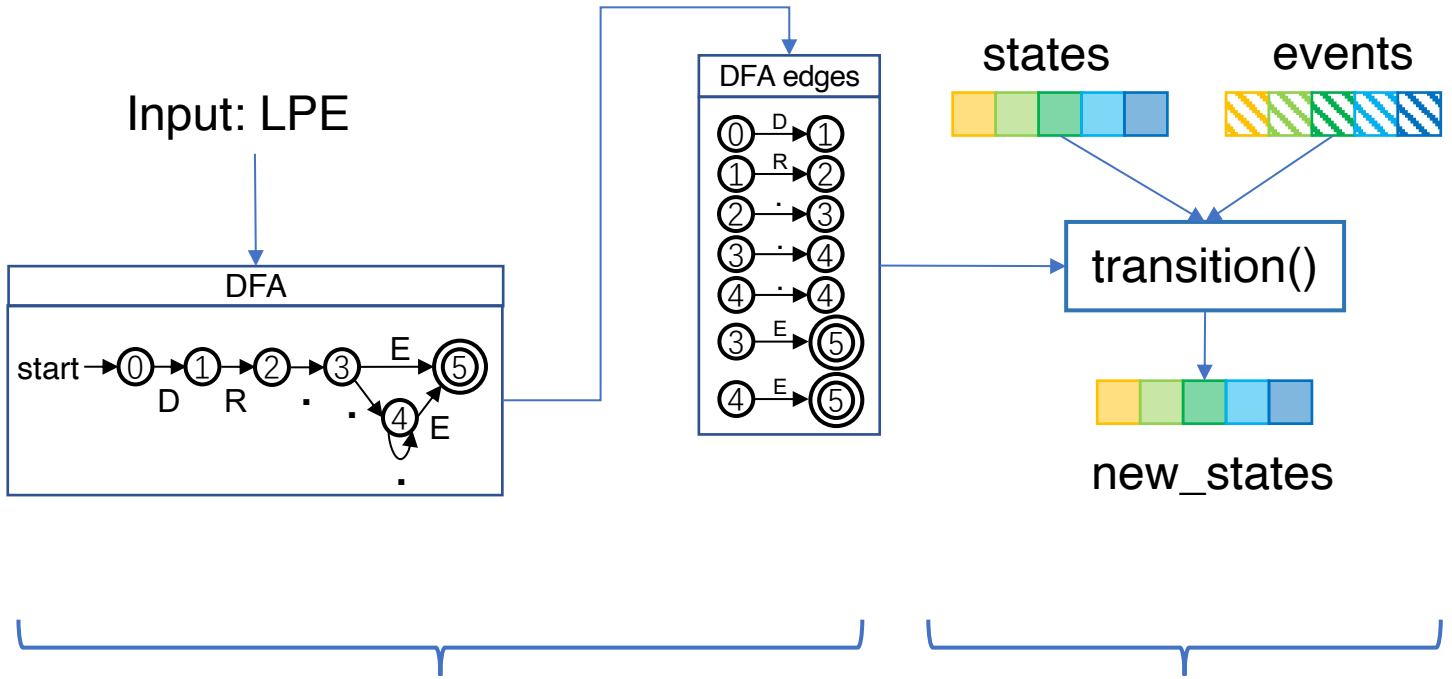
Supported LPE Grammar

- Similar to Pixar and Nvidia LPE renderer grammar
- The precedence from high to low is quantifiers (?, *, +), concatenation, alternatives(|, [])

LPE Events	Description	Note
R	Reflection	
T	Transmission	
D	Diffuse	
G	Glossy	
S	Delta	
E	Emitter	
V	Volume	LPE engine supported. Haven't been validated with scene

Expression	Description	Note
AB	Accepts first A, then B	
A|B	Accepts A or B	
A?	0 or 1 occurrence of A	
A*	0 or more occurrence of A	
A+	1 or more occurrence of A	
.	Any event. A wildcard in any position	
[^A]	Accept not A	LPE engine supported. Haven't been validated with scene
[ABC]	Matches tokens in the alphabet {A,B,C} in any quantity or sequence.	LPE engine supported. Haven't been validated with scene

High Level Design



Path Tracer

Direct emission

call **transition()** to get new states

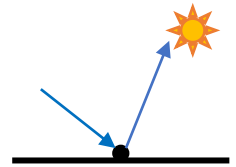
get **accept_mask** from current states

```
Le = dr.select(accept_mask,value,0)
```

Emitter sampling (per lobe NEE)

call **bsdf.pdf()** once for all the lobe with empty bsdf context

```
for lobe in all_lobes:
   # simulate hit bsdf first
   call transition() to get new states
   # then hit an emitter
   call transition() to get new states
   get accept_mask from current states
   call bsdf.eval() to compute bsdf value for individual lobes
   Lr_dir +=dr.select(accept_mask,value,0)
```



```
L = (L + Le + Lr_dir)
```

BSDF sampling

call **transition()** to get new states

get **kill_mask** from current states

```
active = active & ~kill_mask
```

Function Demo

- Input LPE directly

```
inegrator = mi.load_dict(
  {
    'type': 'pmisLPE’,
    'lpe':'ST.*E’,
    'max_depth': 65,
  })
```

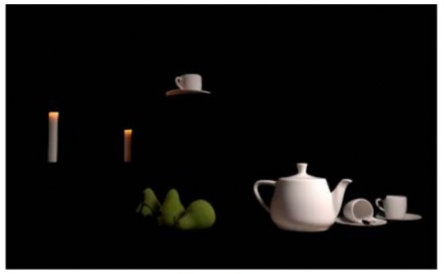
- Get LPE complement

```
inegrator = mi.load_dict(
  {
    'type': 'pmisLPE’,
    'lpe':'ST.*E’,
    'complement': True,
    'max_depth': 65,
  })
```

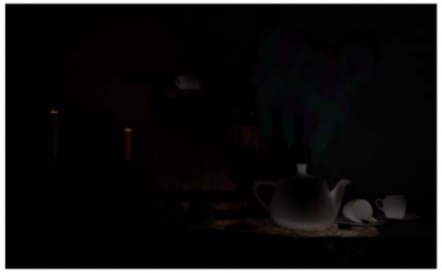
Function Demo

Direct Diffuse

Direct Specular



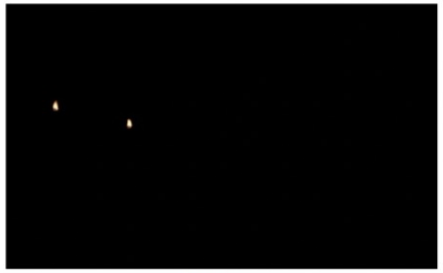
Subsurface



Indirect Diffuse

Indirect Specular

Transmissive



Emissive

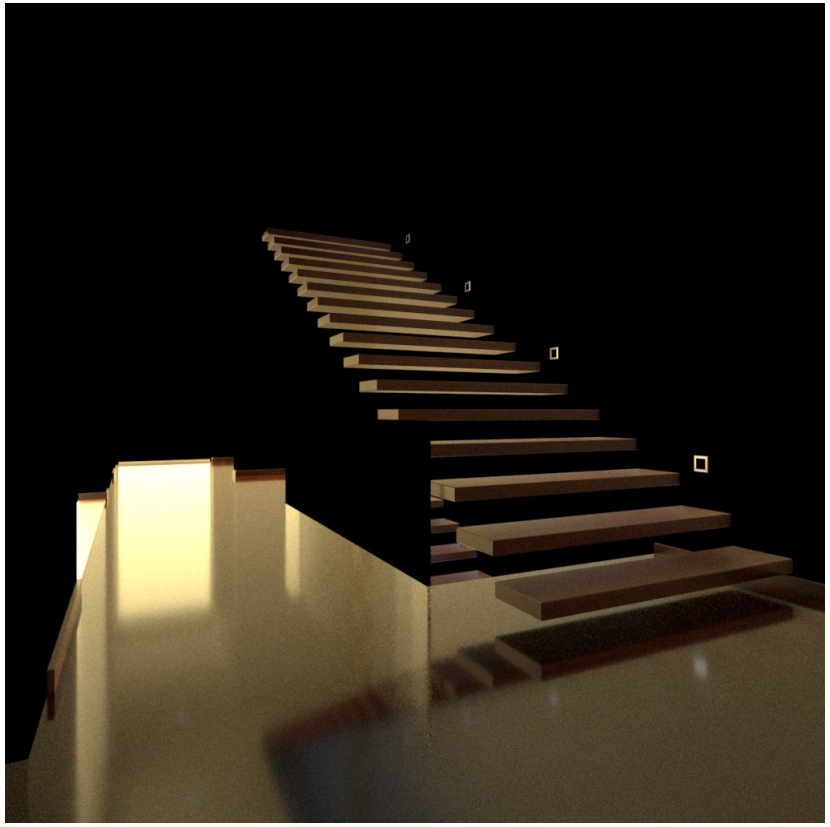
Description	LPE
emissive	E
direct diffuse	DRE
indirect diffuse	DR.+E
subsurface	DT.*E
direct specular	SRE
indirect specular	SR.+E
transmissive	ST.*E

Origin

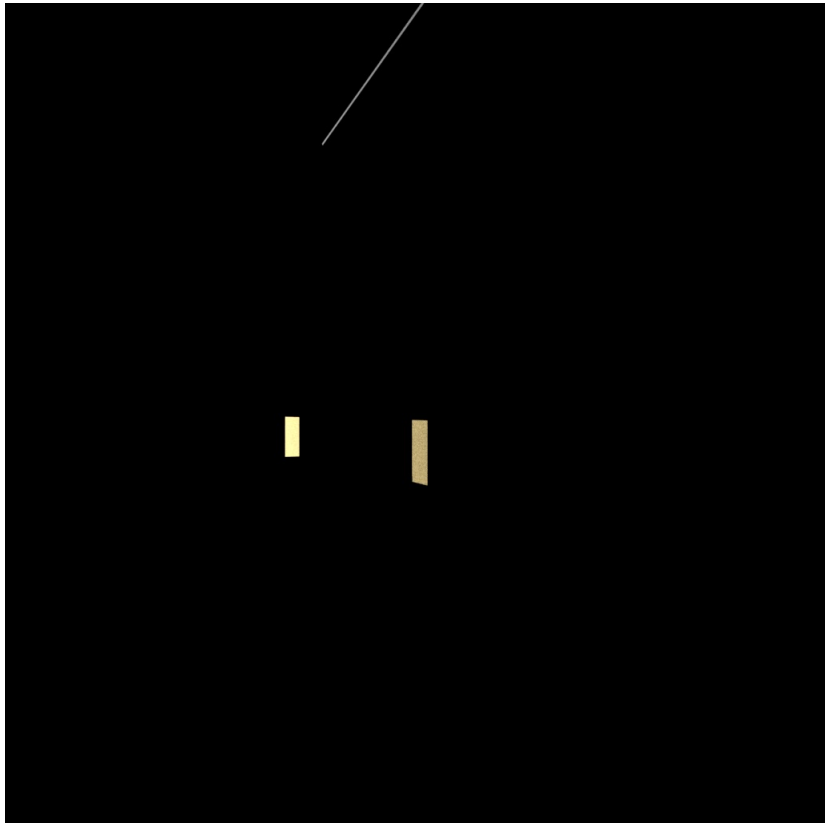
Direct Diffuse

Indirect Diffuse

Glossy

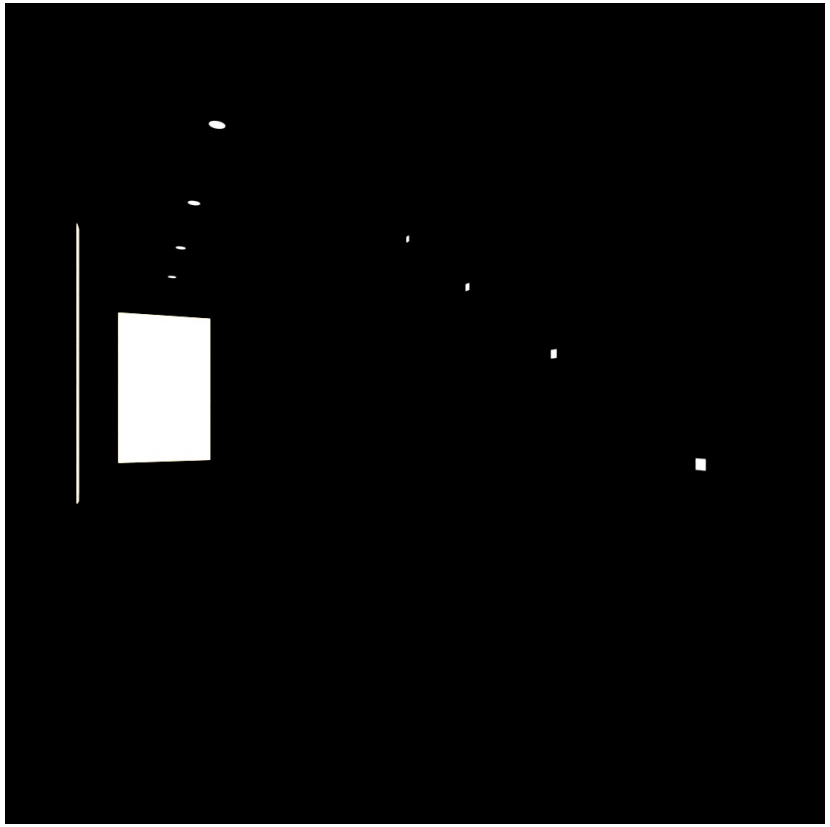


Direct Specular

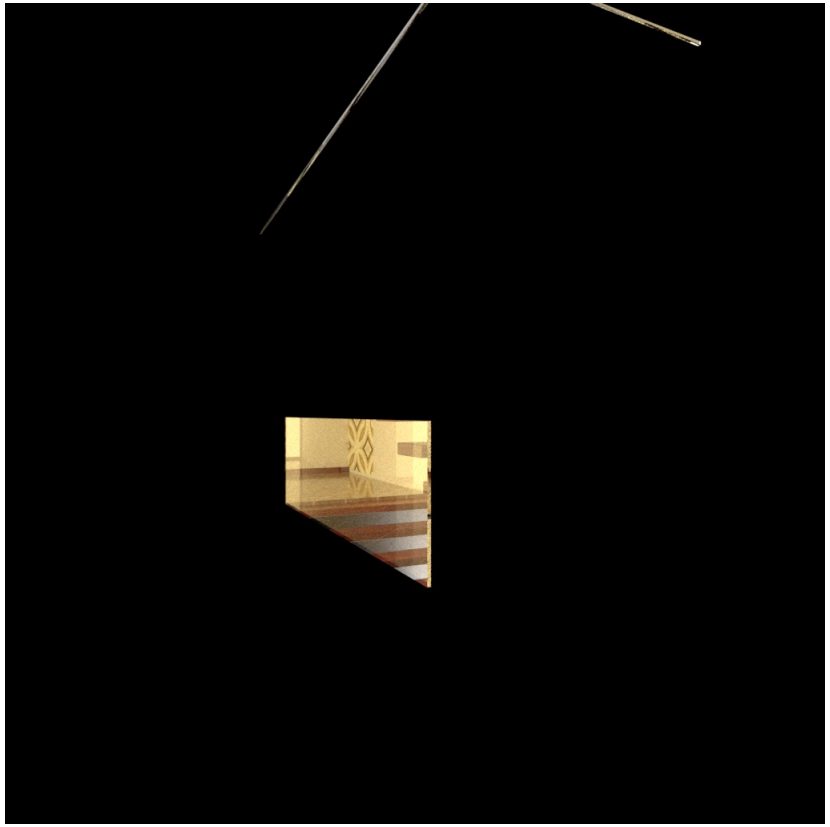


Indirect Specular

Emissive



Transmissive



Transmissive complement

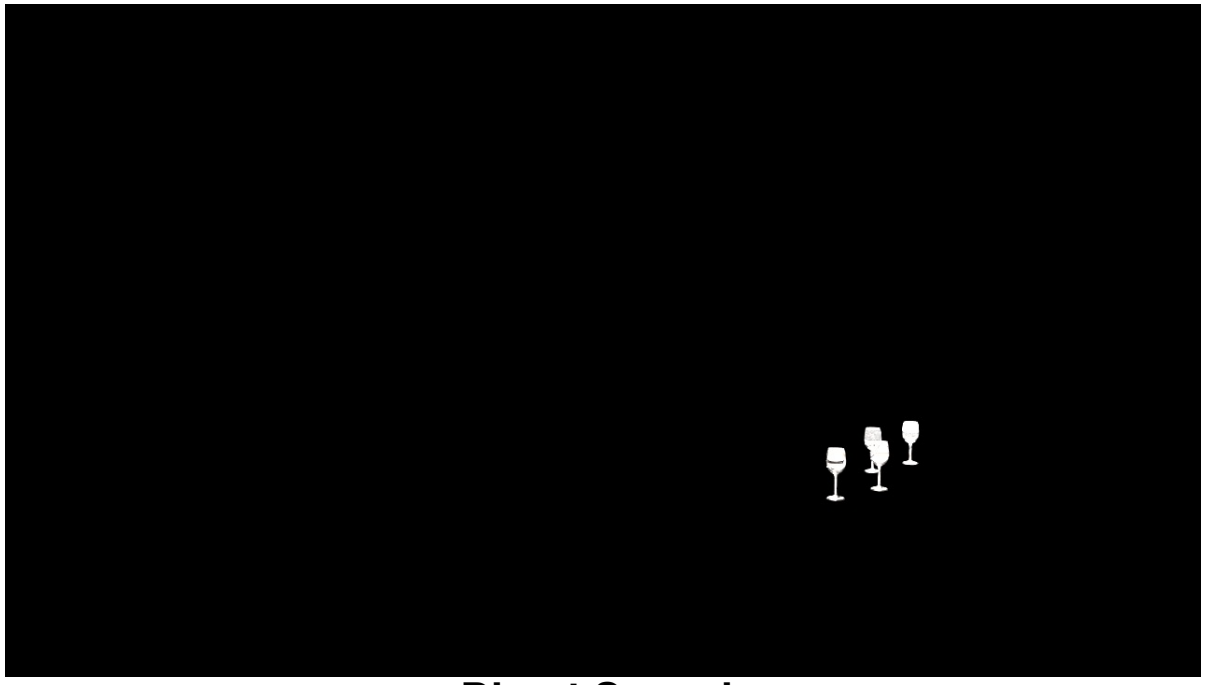
Origin

Emissive

Direct Diffuse

Indirect Diffuse

Transmissive



Glossy

Direct Specular

Indirect Specular

Validation

- Flag validation
  - Compare images filtered by LPE and BSDFFlags
- All-add-up validation
  - Diffuse + Glossy + Specular + Emitter = original image
  - (D.E + G.E + S.E + E) = original image (max depth = 2)
- Complement validation
  - LPE + LPE’s complement = original image

On-going Work

- The current implementation only supports one LPE → one image
  - Therefore in rendering script we need to write many integrators

```
inegrator0 = mi.load_dict(
  {
    'type': 'pmisLPE',
    'lpe':'D.*E',
  })
```

```
inegrator1 = mi.load_dict(
  {
    'type': 'pmisLPE',
    'lpe':'G.*E',
  })
```

```
inegrator2 = mi.load_dict(
  {
    'type': 'pmisLPE',
    'lpe':'S.*E',
  })
```

- Aiming to have
  - Input: multiple LPEs
  - Output: multiple images

```
inegrator = mi.load_dict(
  {
    'type': 'pmisLPE',
    'lpe':['D.*E', 'S.*E', 'G.*E'],
  })
```

Potential Further Work

- The basic idea behind: reduce interference between one another

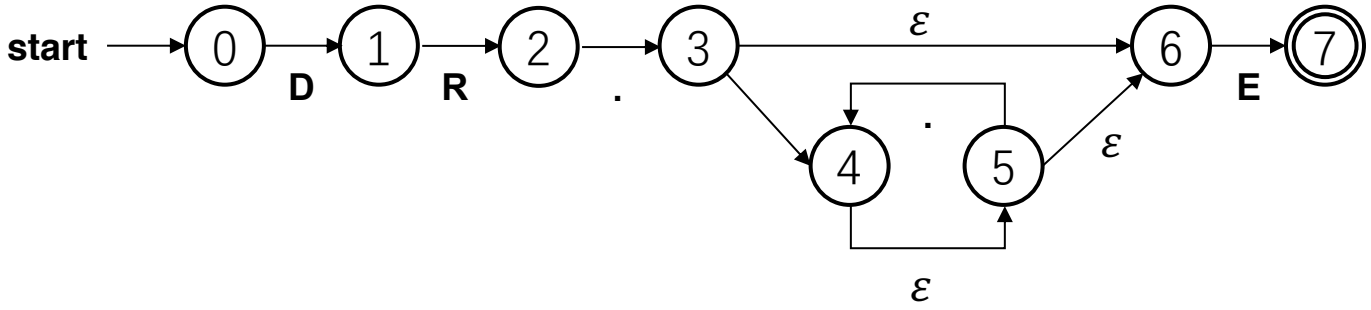
- Differentiable/inverse rendering ←Sébastien
  - Do optimization on different layers(using LPE) of image
  - Potentially lead to better result

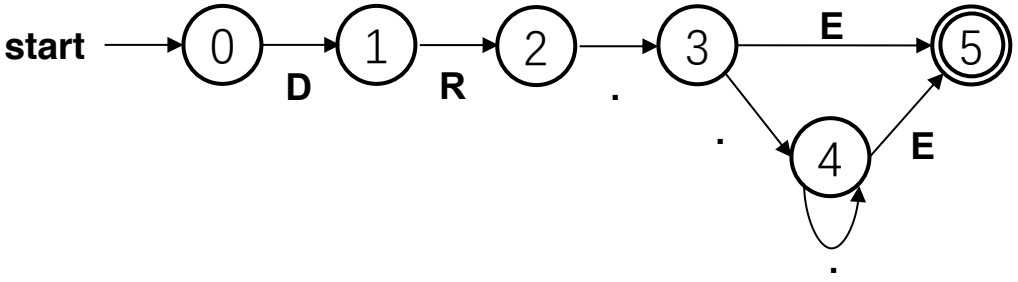
- Denoise ←Nicolas
  - Put multiple layers(using LPE) in neural denoiser, and denoise them individually.
  - Add them all back together to get a final rendering.
  - Might get a better final rendering than just giving a single image to the denoiser.

Supplements

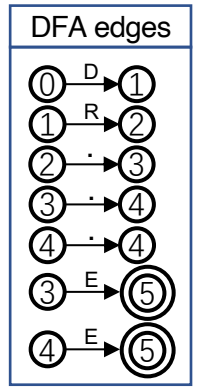
NFA & DFA

- A regular language satisfies the following equivalent properties:
  - it is the language of a regular expression
  - it is the language accepted by a NFA
  - it is the language accepted by a DFA

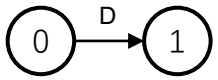




transition(states, events)



- Multi-masked write




```
for e in dfa.edges:
    mask_state = dr.eq(e.origin, states)
    mask_event = dr.eq(e.event, events)
    mask = mask_state & mask_event
    new_states = dr.select(mask, e.nextD, new_states)
# if no matched edges, then set the state to KILLED_STATE
# if the state is in the accept state set, set state to ACCEPT_STATE
```

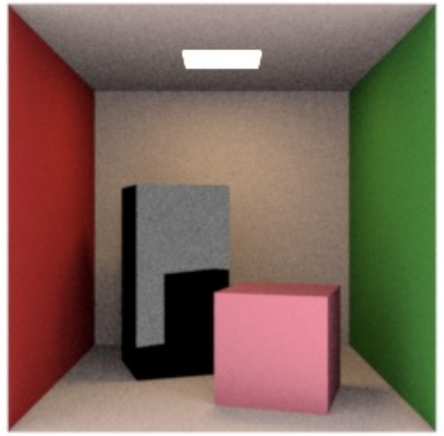
Per lobe NEE

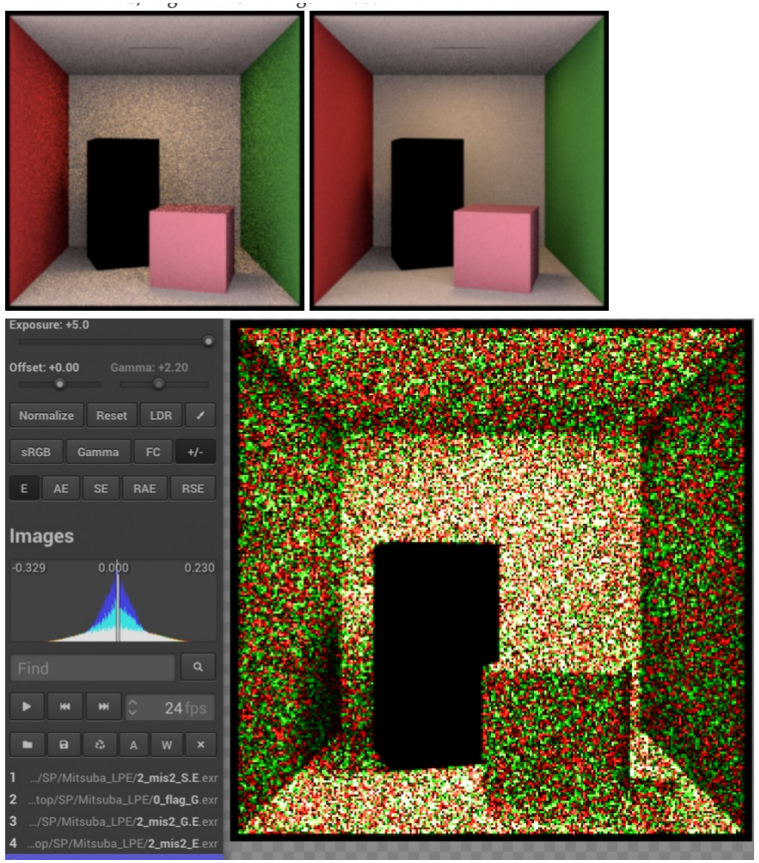
```
Lr_dir = dr.zeros(...)

# call bsdf.pdf() once for all the lobe.
# MIS weight should be computed once only
  outside of the for loop.
empty_ctx = mi.BSDFContext()
bsdf_pdf_em = bsdf.pdf(empty_ctx,...)

mis_em = dr.select(...,
        mis_weight(ds.pdf, bsdf_pdf_em))

# bsdf_flags = [DiffuseReflection,
DiffuseTransmission, GlossyReflection,
GlossyTransmission]
```

```
for flag in bsdf_flags:
   # simulate hit a bsdf first
   # call transition() to get new states
   # then hit an emitter
   # call transition() to get new states
   # get accept_mask from current states

   #call bsdf.eval(temp_ctx, ...) to compute
    the bsdf value for the individual lobes
   temp_ctx = mi.BSDFContext()
   temp_ctx.type_mask = flag
   bsdf_value_em = bsdf.eval(temp_ctx,...)

   Lr_dir += dr.select(temp_accept_mask, value,0)
```

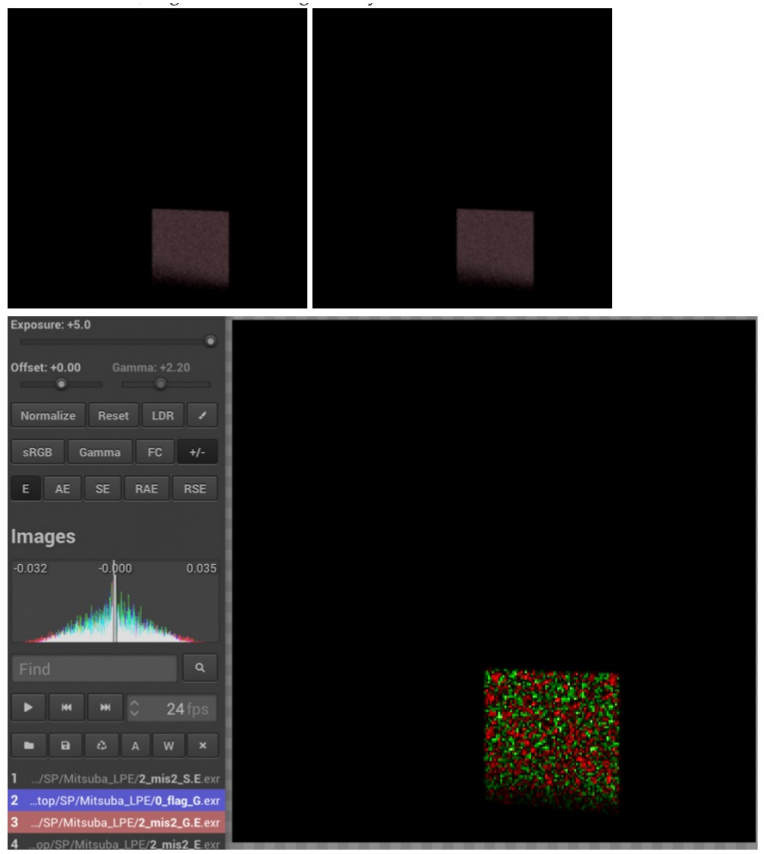
Flag Validation

max_depth=2
Lagre box: dielectric
`BSDFFlags::DeltaReflection`
`BSDFFlags::DeltaReflection`
Small box: roughplastic
`BSDFFlags::GlossyReflection`
`BSDFFlags::DiffuseReflection`

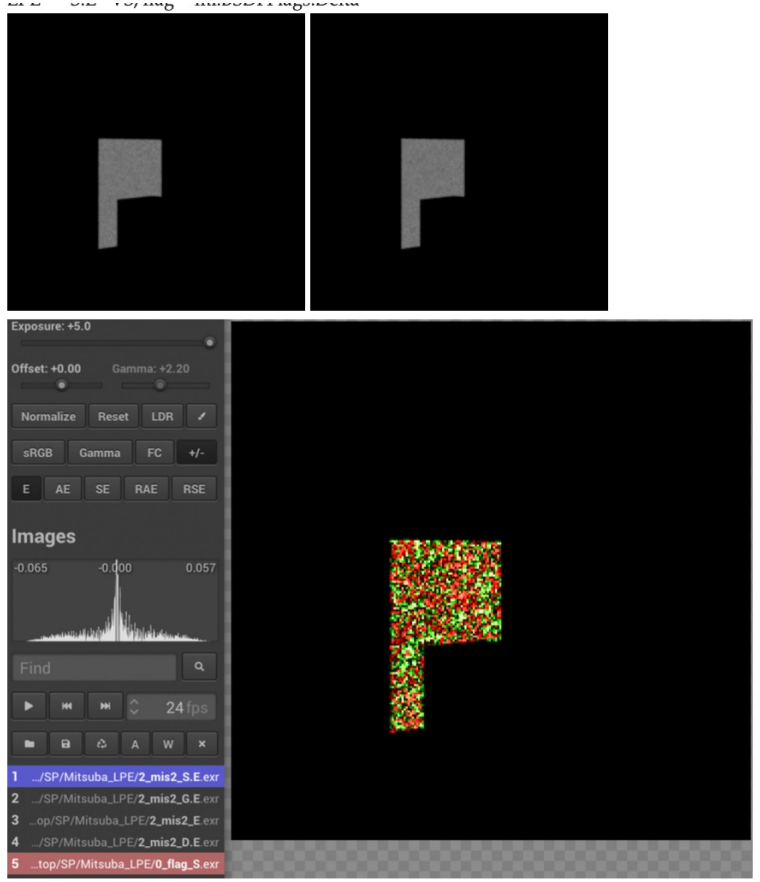




D.E vs BSDFFlags::Diffuse

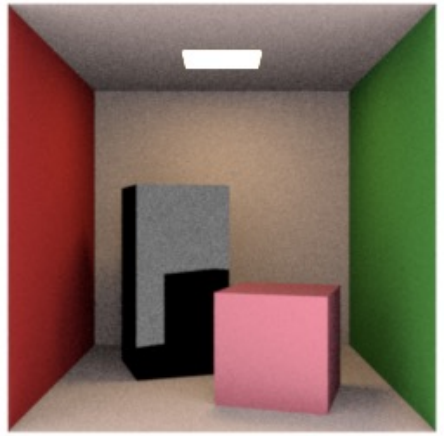


G.E vs BSDFFlags::Glossy

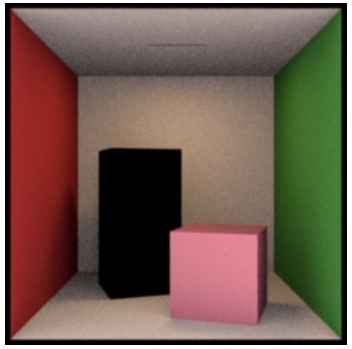


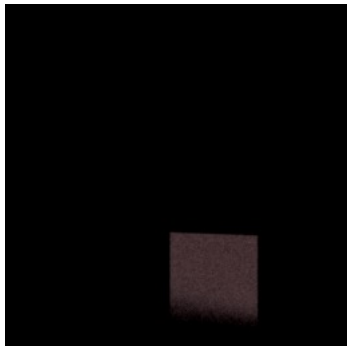
S.E vs BSDFFlags::Delta

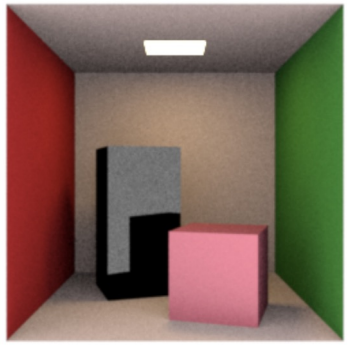
max_depth=2

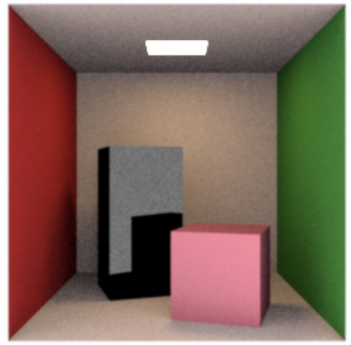


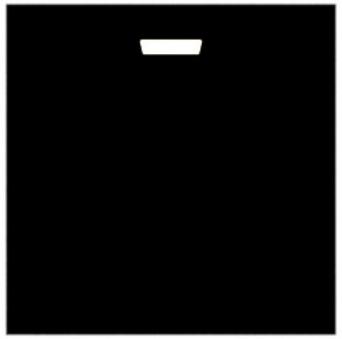
All-add-up Validation

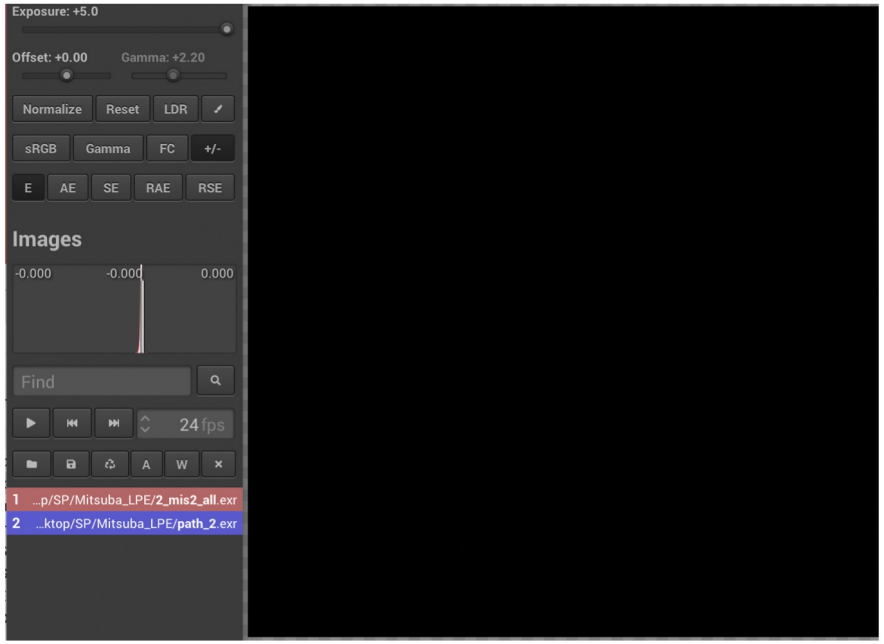






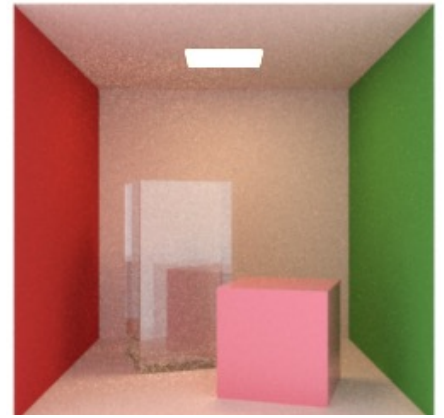


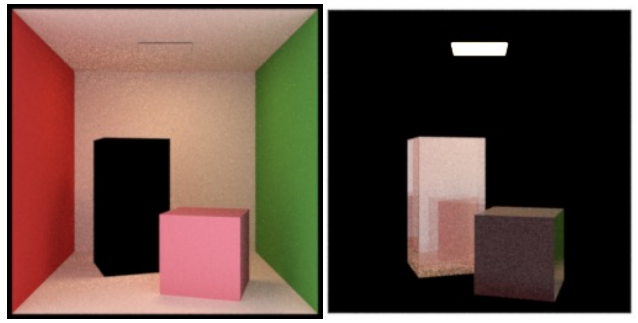


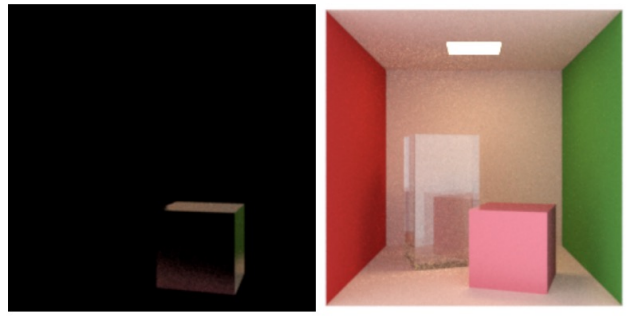


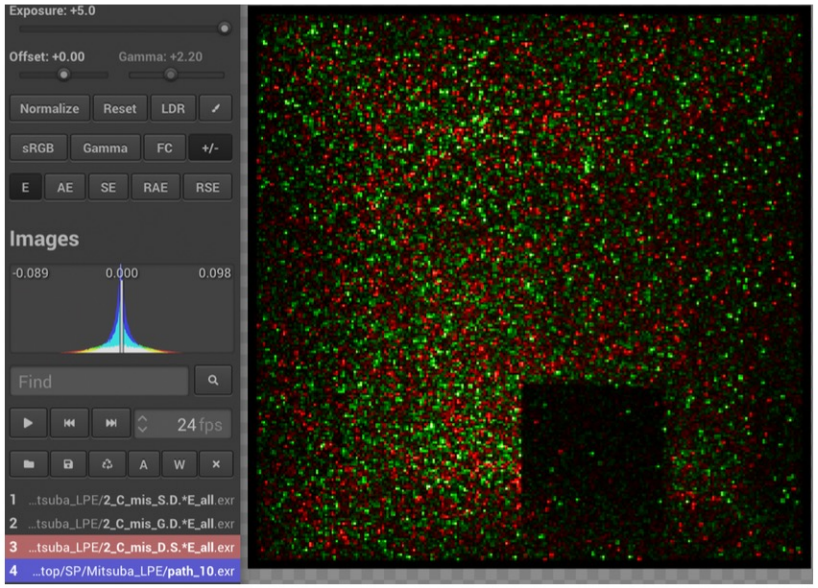
Complement Validation

max_depth=10

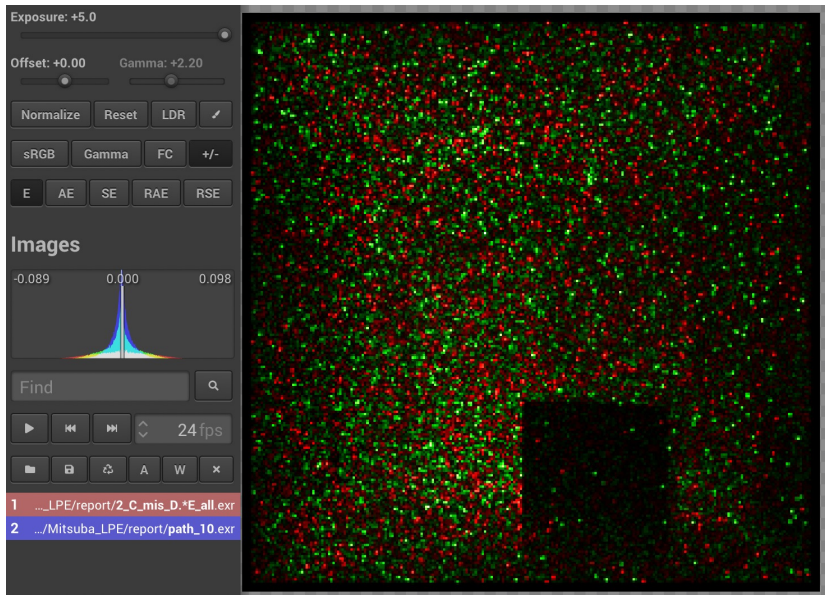




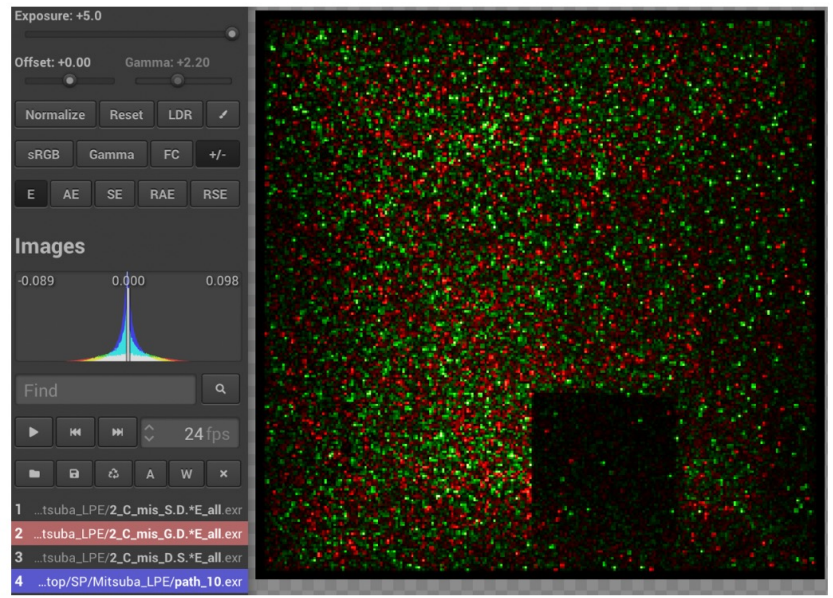




D.S.*E + complement vs Origin



D.*E + complement vs Origin



G.D.*E + complement vs Origin