

# **redqueen** user's manual - simple work flow for everyone

shinji ogaki

July 12, 2016



## Contents

<b>1 About redqueen</b>	<b>3</b>
<b>2 Requisites</b>	<b>3</b>
<b>3 TODO List</b>	<b>3</b>
<b>4 Change Log</b>	<b>4</b>
<b>5 Loading Your Scene</b>	<b>8</b>
5.1 Folder Structure . . . . .	8
<b>6 .rq File Format</b>	<b>8</b>
6.1 Camera . . . . .	9
6.1.1 AOVs . . . . .	10
6.2 Geometry - Object and Instance . . . . .	11
6.2.1 Loading Geometry by .OBJ and .HAIR . . . . .	11
6.2.2 Multi-Level Instancing . . . . .	11
6.3 Render . . . . .	13
6.4 Light . . . . .	14
6.4.1 Point Light . . . . .	14
6.4.2 Parallel Light . . . . .	15
6.4.3 Geometry Light . . . . .	15
6.4.4 Sky Light . . . . .	16
6.4.5 Per Light AOVs . . . . .	17
<b>7 Shader</b>	<b>18</b>
7.1 Surface Shader . . . . .	19
7.1.1 Sideness . . . . .	19
7.1.2 Flags and Options . . . . .	19
7.1.3 Component . . . . .	19
7.1.4 Texture Sampler . . . . .	20
7.1.5 Roughness . . . . .	20
7.2 Surface Shader Examples . . . . .	21
7.2.1 Emitter . . . . .	21

7.3	Volumetric Properties . . . . .	21
7.3.1	Procedural Volumetric Flakes . . . . .	22
7.3.2	Hair Shader . . . . .	23
7.3.3	Random Color Shift . . . . .	24
7.4	Volumetric Properties . . . . .	25
7.5	Geometric Properties . . . . .	26
7.5.1	Options . . . . .	26
7.5.2	Smooth Angle . . . . .	26
7.5.3	Component . . . . .	26
7.6	Image File Formats . . . . .	27
<b>8</b>	<b>Tips for Better Quality</b>	<b>28</b>
<b>9</b>	<b>Using APIs</b>	<b>29</b>
9.1	Rendering Flow . . . . .	29
9.2	Creating Mesh . . . . .	29

## 1 About **redqueen**

- Re-designed from scratch in 2012 aiming at providing the simplest production renderer in the world.
- The goal of this project is to provide a free rendering software for individuals for learning.
- Supports Path Tracing, Progressive Final Gathering (No need to tweak weird parameters), and Photon Mapping.
- All algorithms and shaders work with Multiple Importance Sampling.
- Parallelized with OpenMP 2.0 and supports up to 256 threads.
- All computations are done in single precision.
- Accelerated by 8-ary BVH using AVX and builds a single tree for all types of primitives.
- Über shader is only provided to simplify today's overly complicated work flow.
- Arbitrary per vertex data including multiple uvs is supported. This can be accessed from AOVs.

## 2 Requisites

- CPU that supports AVX instruction sets
- Please install Visual Studio 2015 Community to integrate **redqueen** into your system.

## 3 TODO List

- Python Binding
- Key Frame Data Support
- Faster Motion Blur
- Image Processing
- Mesh Processing
- Reduce Occlusion Tests in Multiple Importance Sampling
- Faster Intersection Test for Hair Primitives
- Better Multiple Importance Sampling for Textured Objects
- Sharing Data with User Applications
- Lens Shader (including Flare)

## 4 Change Log

✗ incompatible + new feature ↗ improvement • bug fix

2016		Class	Details
12 Jul	+ •	shader tree	Illumination computation from non-area lights in participating media Wrong occlusion test results for axis aligned rays
08 Jul	+ •	shader shader	Shadow transparency Overlapped objects having participating media are now handled properly.
07 Jul	+ + ↗ •	shader redqueen tree geometry_light	Different roughness values can be used for the root and tip of a hair strand. Now prints FPS as well as Mrays/s. Ray traversal is 1-2% faster for static objects. Illumination from a sphere was wrong.
06 Jul	↗	tree	BVH construction speed is 10-20% faster.
01 Jul	+ + ✗ ↗	shader shader redqueen geometry	Normal vector perturbation for clear coat Random color shift using instance/object/primitive ids Intel compiler runtime libraries are no longer required. Robust ray sphere intersection test
28 Jun	+ + + ✗ ✗ ✗ ↗ ↗	shader shader shader shader API shader shader shader	Brute-force SSS Clear coat layer Different colors can be used for the root and tip of a hair strand. Simplified shader parameters New APIs for simplified shader parameters Roughness parameter of GGX is squared. Different roughness values can be used for reflected and transmitted rays. Hair shader no longer uses LUTs.
01 Apr	+ + ✗ ↗ ↗	tree shader API tree tree	ATRBVH Procedural volumetric flakes New simpler APIs BVH uses far less memory (up to 50%). Faster multi-level instancing

2015		Class	Details
28 Sep	+	shader	Faster displacement mapping
27 Sep	+ ✗ ↗ ↗ ↗ ↗ ↗	shader shader shader shader tree redqueen	Round corner in <i>geometry_shader</i> for both triangle and tetragon BRDF model switched from Ward with Bounded Albedo to anisotropic GGX. Displacement map uses less run-time memory. Smoothing normal uses less run-time memory. BVH is optimized by agglomerative treelet restructuring. Linked against Freelimage 3.17.0
05 Jul	+	geometry	Shader override for instanced objects
01 Jul	✗	geometry	Hair uvs are automatically assigned only when user specified uvs do not exist.

2015		Class	Details
30 Jun	/	shader redqueen sky_light tree	Textures for displacement mapping is loaded on demand and discarded immediately after use. Better task scheduling Stochastic bilinear filter is used by default to speed up texture mapping. If the number of primitives is over 16.7million, <b>redqueen</b> switches to memory save mode.
27 Jun	/	geometry tree	Automatic hair uv assignment is parallelized. BVH for static objects uses far less memory.
18 Jun	/	tree	Reduced the size of pre-allocated memory for BVH
17 Jun	≠	renderer geometry	distance parameter for secondary final gathering is removed. Now it's calculated automatically. Working memory is reduced when calculating smoothed vertex vectors.
16 Jun	/	tree	Consumes 4 bytes less memory per primitive
12 Jun	/	sky_light renderer sky_light	Smaller LUT for importance sampling Trace depth AOV channel check
11 Jun	•	shader	Artifacts caused by uninitialized uv coord
10 Jun	/	tree geometry geometry	Improved build performance on multi-socket systems Improved hair intersection test AVX2 is not required for hair intersection test.
09 Jun	/	photon_map photon_map geometry geometry shader	Search radius is adaptively changed based on the number of bounces. Each photon consumes 4 bytes less memory. .OBJ loader uses less memory. Artifacts of highly tessellated meshes Very low roughness value caused numerical error.
02 Jun	/	geometry	Each triangle/tetragon consume 4 bytes less memory.
11 May	/	geometry_light sky_light API	Increased accuracy in MIS computation Increased accuracy in MIS computation Display gamma was broken
09 May	+	tree tree tree tree tree tree tree AOV	Improved version of "Child Node Sorting for Fast Occlusion Test" A slightly smaller memory footprint Faster occlusion test Robust intersection test for multi-level instancing Crash when instanced object refers to non-existing object Crash when empty object is used Camera AOVs are broken due to the change of APIs.
04 May	•	shader	Fixed fresnel term for <i>thin_dielectric</i>
02 May	≠	tree shader	Occasional crash due to insufficient stack size Clip map is removed and opacity map is implemented.
30 Apr	+	renderer sky_light	Color of error pixels (black by default) Improved stratification and initialization
26 Apr	/	image sky_light shader	Date and time is added to the file name of beauty. Better stratification Smoothing normals
21 Apr	•	shader	Minor bug in shadow transparency
20 Apr	+	geometry	Runtime tessellation of hair
19 Apr	≠	API	New simplified APIs
04 Jan	+	geometry sampler	<i>triangle_texture</i> is added. Stochastic bilinear filter for image sampling
03 Jan	+	geometry API	<i>instance</i> is added to support multi-level instancing. new APIs to create objects
01 Jan	+	tree geometry_light parallel_light geometry	Multi-level instancing with motion blur Minor bug in sampling Minor bug in photon casting Robust planar check for tetragons



2013		Class	Details
15 Dec	↗ ● ● ●	geometry geometry_light geometry shader	Faster initialization (the "loading object" part) Tetragon sampling issue Better ray-cylinder intersection test Edge darkening when using glossy material with normal map
10 Dec	↗ ● ● ●	geometry tree geometry geometry	.OBJ loader is 30% faster. Missing objects when rendering gigantic data sets Packed vectors now use 32bits not 16bits. This slows down a bit. Degenerated tetragon removal
07 Dec	↗ ↗	photon_map photon_map	nearly 200% faster photon gathering for a large search radius photon mapping uses the offset option of non-area lights
05 Dec	✚ ↗ ●	API API photon_map	Functions to load multiple particles/cylinders/triangles/tetragons at once Switched from static lib to dynamic lib. Dark caustics issue
31 Nov	✗ ✚ ●	render geometry shader	Using negative values skips clamping. Convex regular tetragons is supported as an atomic primitive. High glossy materials render black.
25 Nov	↗ ↗ ↗ ●	tree tree tree geometry	Tree construction is 10% faster. Tree node uses 2/3 memory. Faster geometry loading Degenerated triangle removal
22 Nov	✚ ↗	redqueen tree	Created Visual Studio 2013 version, which is slightly faster. Peak memory consumption during tree construction is greatly reduced.
14 Nov	✗ ✚ ✚ ✚ ✚ ✚ ✚ ✚ ●	shader shader point_light point_light parallel_light geometry geometry geometry	angle map uses $[0, 1]$ normalized value (mapped to $[0, 2\pi]$ ) not degree. Layerable surface_shader with weight control Spot light offset is added to prevent jagged shadows. offset is added to prevent jagged shadows. Sampling huge environment maps (8k, 16k, ...) is done in nearly constant time. Better importance sampling of textured triangular meshes Density Estimator uses less memory.
07 Nov	✗ ✚ ✚	geometry geometry redqueen	Shader assignment should be done by "usemtl" not by group "g". Hair/Fur is now activated. Initial version of rqAPIs( rq.lib(x64) and rq.h )
29 Oct	↗	redqueen	3% speed up
28 Oct	✚ ✚	camera geometry	Render region Faster mesh loading via textures
26 Oct	● ● ●	sky_light redqueen documentation	MIS has been broken since SSS was implemented. A proper build date by the macro of Visual Studio. Discarded an un-professional document. Preparing a little bit better tex manual.
25 Oct	✗ ✚ ✚ ✚ ✚	display shader geometry shader triangle	The default image output file format is now EXR. Image Sampler: UV matrix, color matrix, and gamma Particle generation from RGBA(x, y, z, radius) textures Ray Traced-SSS Accurate area size computation for robust illumination computation
11 Oct	✚ ↗ ● ●	geometry tree shader geometry_light	Particle is now activated. Faster .OBJ loader (50% speedup for single thread and more for multi-thread) Smoothing normal issue Sampling issue
10 Oct	✚ ↗ ● ●	shader tree shader geometry_light	Ray-Traced SSS is now activated. Fast breadth-first tree construction (from $\times 2$ to $\times 3$ faster) Improved sampling number control when using car paint-like shaders. Weird lighting patterns caused by geometry_light
06 Oct	●	shader	Density estimator and MIS

## 5 Loading Your Scene

### 5.1 Folder Structure

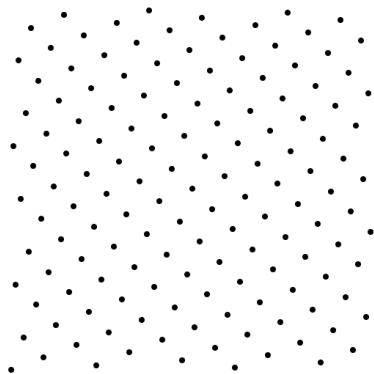
**redqueen** takes the name of folder that contains your scene as its argument. The folder should be organized as follows for easy asset management.

```
your_folder--|
    |--camera    (contains .rq files for cameras)
    |--geometry  (contains .rq files for geometry, OBJs, etc.)
    |--light     (contains .rq files for lights, environment map, etc.)
    |--render    (contains .rq files for render setting)
    |--shader    (contains .rq files for shaders, textures, etc.)
```

## 6 .rq File Format

Basically, you can write anything in any file in any order as **redqueen** reads everything and store them in memory, and then initializes. **Keywords written in bold** are layerable or can be written multiple times.

## 6.1 Camera



Multiple camera poses will be supported in the future. For now, 2 poses are required for shutter start and shutter end. All parameters are linearly interpolated when you use camera blur. *sample* in the *camera* affects the quality of anti-aliasing and DOF. Other renderers use  $1 \times 1$ ,  $2 \times 2$ ,  $3 \times 3$ ... On the other hand, **redqueen** uses a single integer so that more flexible quality control is possible. This is realized by using fibonacci lattice (the left image).

<b>camera</b>		<i>unit</i>	
name	"nikon"	<i>string</i>	camera name
resolution	1024 1024	2 <i>integers</i>	sensor resolution, image size
region	5 5 90 90	4 <i>integers</i>	render region, left top and right bottom positions
sample	17	<i>integer</i>	anti aliasing
exposure	1 1 1	<i>rgb</i>	exposure
projection	"perspective"	<i>string</i>	projection type
<b>aov</b>			options for per light AOVs
<b>type</b>	"normal"	<i>string</i>	component to write out
<b>type</b>	"arbitrary_name"	<i>string</i>	user data to write out e.g. <i>pref</i> and <i>motion_blur</i>
image	"normal.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>pose</b>			camera pose for shutter open
position	0 2 4	<i>xyz</i>	start position
target	0 2 0.5	<i>xyz</i>	target point
up_vector	0 1 0	<i>xyz</i>	up vector
bokeh	0	<i>degree</i>	blur angle at the target point
field_of_view	87	<i>degree</i>	fov
time	0	<i>scalar</i>	[0,1] normalized value, ignored now
<b>end</b>			
<b>pose</b>			camera pose for shutter close
position	0 2 4	<i>xyz</i>	end position
target	0 2 0.5	<i>xyz</i>	target point
up_vector	0 1 0	<i>xyz</i>	up vector
bokeh	0	<i>degree</i>	blur angle at the target point
field_of_view	87	<i>degree</i>	fov
time	1	<i>scalar</i>	[0,1] normalized value, ignored now
<b>end</b>			
<b>end</b>			

## References

*Spherical Fibonacci Point Sets for Illumination Integrals*

### 6.1.1 AOVs

Supported AOV types are: *lambertian*(=*diffuse*), *glossy*, *specular*, *emission*, *normal*, *tangent*, *depth*, and *position*. Users can write per vertex data into an image.

```
rqAddVertexData ( object_id, part_id, "my_data", n, dimension, data );
```

If you add the above data for vertices, they can be saved as images by writing as follows.

```
camera
  aov
    image "user_color.png"
    type "lambertian"
    type "glossy"
  end
  aov
    image "user_color.png"
    type "color"
  end
  aov
    image "uv_layer1.png"
    type "uv1"
  end
  aov
    image "pref.png"
    type "reference"
  end
  aov
    image "motion_vector.png"
    type "motion"
  end
  ...
end
```

Each *part* has several vertex data as default: position, uv0, normal, tangent, and radius. These parameters can also be saved in the same way. Multiple illumination components such as *lambertian* and *glossy* can be saved in a single image. However, you cannot save multiple user data in a single file because, for example, mixing position and texture uv is meaningless.

## 6.2 Geometry - Object and Instance

```
geometry--|
    |--table.rq (refers table.obj)
    |--car.rq (refers car.obj)
    |--house.rq (refers house.obj)
    |--animal.rq (refers animal.obj)
```

### 6.2.1 Loading Geometry by .OBJ and .HAIR

Exporting one gigantic OBJ file is not recommended. Using many small OBJ files leads to less memory consumption. **redqueen** can render hair strands. Hair strands can be loaded by using .HAIR file format as follows. Currently each hair segment is represented by a truncated cone. The HAIR models are available at <http://www.cemyuksel.com/research/hairmodels/>.

```
object
    name    "woman"
    shader  "undefined"
    obj     "model\Hair\geometry\woman.obj"
    hair    "model\Hair\geometry\wCurly.hair"
    matrix  1 0 0 0    0 1 0 0    0 0 1 0    0 0 0 1
end
```

### 6.2.2 Multi-Level Instancing



<b>instance</b>		<i>unit</i>	
object	"name"	<i>string</i>	object name
shader	"plastic"	<i>string</i>	shader to assign
matrix	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1	<i>4x4matrix</i>	a b c d e f g h i j k l m n o p
end			

**redqueen** supports multi-level instancing (up to 16 levels). Every *object* can have instanced *objects* but they should not refer their parent *object*. Non-referred *object* becomes the root *object*. Scale, rotate, and translate are supported. Shear is not supported yet. Photon casting works fine for instanced light emissive objects.

```

object
    name "tree"
    obj  "model\Instance\geometry\tree.obj"
end

object
    name "forest"
    instance
        matrix 1 0 0 -2      0 1 0 0      0 0 1 -2      0 0 0 1
        object "tree"
    end
    instance
        matrix 1 0 0 2      0 1 0 0      0 0 1 -2      0 0 0 1
        object "tree"
    end
    instance
        matrix 1 0 0 2      0 1 0 0      0 0 1 2      0 0 0 1
        object "tree"
    end
    instance
        matrix 1 0 0 -2      0 1 0 0      0 0 1 2      0 0 0 1
        object "tree"
    end
end

object
    name "mountain"
    instance
        matrix 1 0 0 -1      0 1 0 2      0 0 1 -1      0 0 0 1
        object "forest"
    end
    instance
        matrix 1 0 0 -1      0 1 0 2      0 0 1 1      0 0 0 1
        object "forest"
    end
    instance
        matrix 1 0 0 1       0 1 0 2      0 0 1 -1      0 0 0 1
        object "forest"
    end
    instance
        matrix 1 0 0 1       0 1 0 2      0 0 1 1      0 0 0 1
        object "forest"
    end
end

```

## References

- Fast Parallel Construction of High-Quality Bounding Volume Hierarchies*
- Faster Incoherent Ray Traversal Using 8-Wide AVX Instructions*
- Getting Rid of Packets*
- MBVH Child Node Sorting for Fast Occlusion Test*

### 6.3 Render

**redqueen** supports 2 rendering modes: 1. pure path tracing and 2. progressive final gathering. The setting for the both methods are described in the final\_gather sub section.

render			<i>unit</i>	
clamp	1 1 1		<i>rgb</i>	clamping to get rid of fire flies
gamma	2.2		<i>scalar</i>	display gamma
error	100 0 0		<i>rgb</i>	color for sub sample that has numerical error
<b>image_path</b>	"..../images"		<i>string</i>	image file search path
<b>image_path</b>	"..../hdr"		<i>string</i>	image file search path
final_gather				options for the FG integrator
sample	256		<i>integer</i>	indirect GI samples for both <i>lambertian</i> and <i>glossy</i>
ray_depth	8		<i>integer</i>	ray trace depth ( eye path )
photon_depth	8		<i>integer</i>	photon trace depth ( light path )
bounce	2		<i>integer</i>	GI bounce both for <i>lambertian</i> and <i>glossy</i>
radius	0.1		<i>m</i>	photon search radius
resolution	0.05		<i>m<sup>2</sup></i>	density estimation resolution
end				
end				



Figure 1: Without secondary final gathering, image has artifacts (left). **redqueen** does not have this problem as it uses progressive final gathering and automatically calculates the distance for secondary final gathering.

### References

- The Rendering Equation*
- Global Illumination Compendium*
- Realistic Image Synthesis Using Photon Mapping*

## 6.4 Light

### 6.4.1 Point Light



Figure 2: By setting *offset* 0.1, the jagged shadow problem on the sphere is solved.

<b>point_light</b>		<i>unit</i>	
color	1 1 1	<i>rgb</i>	the intensity of light
position	1 1 1	<i>xyz</i>	position of light
direction	1 1 1	<i>xyz</i>	direction of spot light
offset	0.05	<i>scalar</i>	to prevent jagged shadow edge, ranges from 0 to 1
inner_angle	10	<i>degree</i>	fall off start angle
outer_angle	30	<i>degree</i>	fall off end angle
photon	1000000	<i>integer</i>	the number of photons to shoot
sample	16	<i>integer</i>	the number of samples for shadow rays and direct BSSRDF
blur	1	<i>m</i>	jitter radius
<b>aov</b>			options for per light AOVs
<b>type</b>	"lambertian"	<i>string</i>	component to write out
image	"lambertian.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>link</b>			options for light link
<b>shader</b>	"shader_name"	<i>string</i>	the name of shader
<b>end</b>			
<b>end</b>			

If you set values for *inner\_angle*, *outer\_angle*, and *direction*, *point\_light* also works as a spot light. When using *offset*, intensities are computed as

$$I_{lambertian} = (+N \cdot L - offset)/(1 - offset) \quad (1)$$

$$I_{transmittance} = (-N \cdot L - offset)/(1 - offset), \quad (2)$$

where  $N$  is a normal vector and  $L$  is the direction of a shadow ray. More intuitively, *offset* narrows down illuminated regions, and hence introduces bias.

## 6.4.2 Parallel Light

<b>parallel_light</b>		<i>unit</i>	
color	1 1 1	<i>rgb</i>	the intensity of light
direction	1 1 1	<i>xyz</i>	direction of light
offset	0.05	<i>scalar</i>	to prevent jagged shadow edge, ranges from 0 to 1
photon	1000000	<i>integer</i>	the number of photons to shoot
sample	64	<i>integer</i>	the number of samples for shadow rays and direct BSSRDF
blur	0.5	<i>degree</i>	blur angle
<b>aov</b>			options for per light AOVs
<b>type</b>	"specular"	<i>string</i>	component to write out
<b>type</b>	"glossy"	<i>string</i>	component to write out
image	"spec_glossy.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>link</b>			options for light link
<b>shader</b>	"shader_name"	<i>string</i>	the name of shader
<b>end</b>			
<b>end</b>			

## 6.4.3 Geometry Light

This light is different from the previous 2 lights. Geometry light is a singleton and works with polygons that has a shader with non-zero *emission* and *virtual\_light* option *on*. Geometry Light supports Multiple Importance Sampling. If *sample* is greater than 0 and there exists an *object* with the *virtual\_light* option *on*, MIS works automatically.

<b>geometry_light</b>		<i>unit</i>	
photon	100000	<i>integer</i>	the number of photons to shoot
sample	128	<i>integer</i>	the number of samples for shadow rays and direct BSSRDF
<b>aov</b>			options for per light AOVs
<b>shader</b>	"red_light"	<i>string</i>	the name of a shader with an emission component
<b>type</b>	"emission"	<i>string</i>	component you want write out
<b>type</b>	"specular"	<i>string</i>	component you want write out
<b>type</b>	"lambertian"	<i>string</i>	component you want write out
<b>type</b>	"glossy"	<i>string</i>	component you want write out
image	"red_light.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>aov</b>			options for per light AOVs
<b>shader</b>	"green_light"	<i>string</i>	the name of a shader with an emission component
<b>type</b>	"emission"	<i>string</i>	component to write out
<b>type</b>	"specular"	<i>string</i>	component to write out
<b>type</b>	"lambertian"	<i>string</i>	component to write out
<b>type</b>	"glossy"	<i>string</i>	component to write out
image	"green_light.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>link</b>			options for light link
<b>shader</b>	"shader_name"	<i>string</i>	the name of shader
<b>end</b>			
<b>end</b>			

## References

- Robust Monte Carlo Methods for Light Transport Simulation*
- Fast Random Sampling of Triangular Meshes*

#### 6.4.4 Sky Light

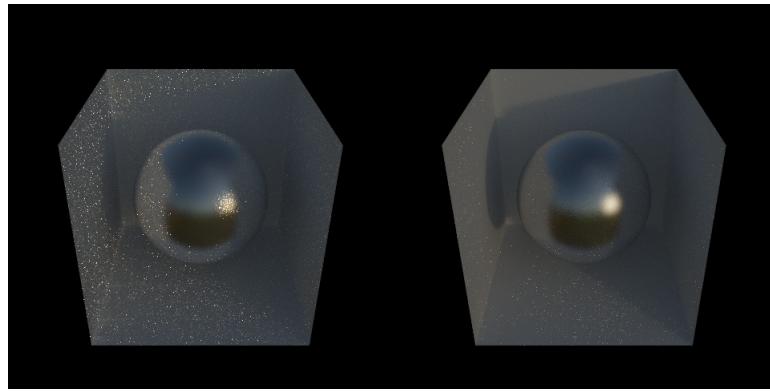


Figure 3: *sky\_light* MIS. The left image was rendered without MIS and is missing the illumination from the sun. The average pixel values are 0.496 for the left image and 0.497 for the right image, respectively.

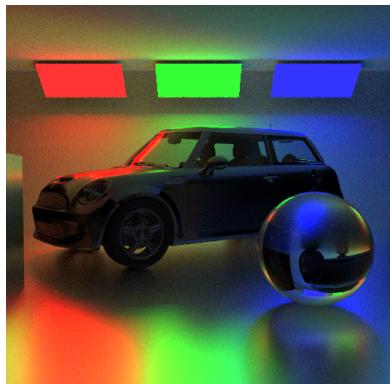
Sky Light supports Multiple Importance Sampling. If *sample* is greater than 0 and an environment map is loaded, MIS works automatically.

<b>sky_light</b>		<i>unit</i>	
photon	100000	<i>integer</i>	the number of photons to shoot
sample	128	<i>integer</i>	the number of samples for shadow rays and direct BSSRDF
image	"env.exr"	<i>string</i>	texture used for Image Based Lighting
backdrop	"black.exr"	<i>string</i>	texture for backdrop/primary visibility
color	1 1 1	<i>rgb</i>	the intensity of sky or multiplier for the environment map
north	0 0 1	<i>xyz</i>	the direction to north
zenith	0 1 0	<i>xyz</i>	the direction to zenith
<b>aov</b>			options for per light AOVs
<b>type</b>	"lambertian"	<i>string</i>	component to write out
image	"transmissive.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>link</b>			options for light link
<b>shader</b>	"shader_name"	<i>string</i>	the name of shader
<b>end</b>			
<b>end</b>			

#### References

- Robust Monte Carlo Methods for Light Transport Simulation*
- Monte Carlo Rendering with Natural Illumination*
- Fast Random Sampling of Triangular Meshes*
- Real-time KD-Tree Based Importance Sampling of Environment Maps*

#### 6.4.5 Per Light AOVs



**redqueen's** per light AOVs work in the same way as the Multilight (<http://support.nextlimit.com/display/mxdocsv3/Multilight>). You can also save different components (lambertian, glossy, and specular) as different files.



Figure 4: Per light AOVs.



Figure 5: Saving different components of the green light.

## 7 Shader

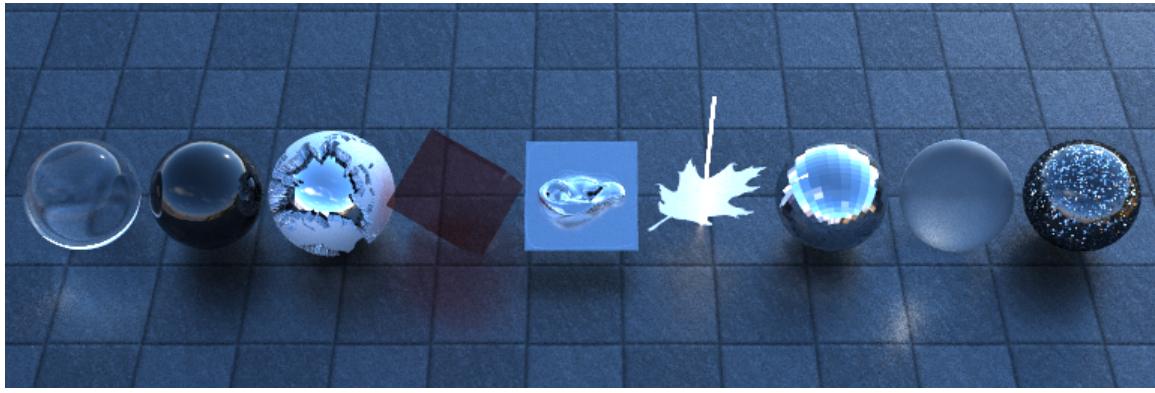


Figure 6: Material examples.

This section describes how to use **redqueen**'s Über shader. This is the most complicated part of **redqueen**.

<b>shader</b>	<i>unit</i>
single_sided	
<b>component</b>	
<b>end</b>	
geometry	
<b>end</b>	
volume	
<b>end</b>	
<b>end</b>	

## 7.1 Surface Shader

<i>name_of_component</i>		<i>unit</i>	the type of component
side		<i>string</i>	sideness: "outer"="face", "inner"="back", and "both". you may also write sideness instead of side.
color	1 1 1	<i>rgb</i>	color or multiplier for texture
image	"texture.png"	<i>string</i>	the name of texture map
level	1	<i>integer</i>	not used in <i>surface_shaders</i> but in <i>geometry_shaders</i>
sampler			texture sampler
end			
end			

If the *single\_sided* option is used, the "face" colors are applied for the both front and back sides of a polygon.

### 7.1.1 Sideness

The side options are: "face"("outer"), "back"("inner"), and "both". You can also use "root" and "tip" to specify different colors for the root and tip of a hair strand.

### 7.1.2 Flags and Options

The currently supported flags are: *camera\_primary*, *light\_primary*, *virtual\_light*, and *invisible*.

### 7.1.3 Component

**redqueen** supports the following components: *diffuse*, *glossy*, *specular*, *emission*, *roughness*, *normal*, *shadow*, *clear\_coat*, and *clear\_coat\_ior*. All components have the same parameters so it is easy to use. *glossy* should be used together with *roughness*. *clear\_coat* is a layer to simulate a top clear coat and interpreted as a specular layer. *shadow* can be used to control shadow transparency.

### References

*Bounding the Albedo of the Ward Reflectance Model*

*Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs*

*Importance Sampling Microfacet-Based BSDFs using the Distribution of Visible Normals*

#### 7.1.4 Texture Sampler

sampler		<i>unit</i>	
uv_matrix	1 0 0 0 1 0 0 0 1	3x3matrix	a b c d e f g h i
color_matrix	1 0 0 0 0 1 0 0 0 1 0 0 0 0 1	4x4matrix	a b c d e f g h i j k l m n o p
gamma	2.2	scalar	gamma correction
pixel_sampler	"stochastic"	string	pixel sampler
end			

*pixel\_sampler* can be chosen from "nearest", "stochastic" (stochastic bilinear filter), and "bilinear". The execution of order of the above 3 conversions is: 1. UV matrix conversion, 2. gamma correction, and 3. color matrix conversion. The transformed texture coordinate ( $U'$ ,  $V'$ ,  $W' = 1$ ) is given as follows. The input texture coordinate is expanded to homogeneous coordinate so that translation, scaling, and rotation can be done with one matrix multiplication. Normally  $g = h = 0$  and  $i = 1$ , and  $c$  and  $f$  are for translation.

$$U' = (aU + bV + cW)/(gU + hV + iW) \quad (3)$$

$$V' = (dU + eV + fW)/(gU + hV + iW) \quad (4)$$

$$W' = (gU + hV + iW)/(gU + hV + iW). \quad (5)$$

As you may have noticed, the color is converted in the same way. Normally  $m = n = o = 0$  and  $p = 1$ .

$$R' = (aR + bG + cB + dE)/(mR + nG + oB + pE) \quad (6)$$

$$G' = (eR + fG + gB + hE)/(mR + nG + oB + pE) \quad (7)$$

$$B' = (iR + jG + kB + lE)/(mR + nG + oB + pE) \quad (8)$$

$$E' = (mR + nG + oB + pE)/(mR + nG + oB + pE). \quad (9)$$

#### 7.1.5 Roughness



Figure 7: Roughness. From left to right: 1, 0.5, 0.25, 0.125, and 0.0625

The above image shows how *roughness* parameter affects the glossiness.

## 7.2 Surface Shader Examples

### 7.2.1 Emitter



Figure 8: *geometry\_light* MIS. The left image does not use *virtual\_light*. The center image was rendered *virtual\_light* enabled. Using clamp generates clean image (right).

**redqueen** automatically generates many virtual light sources on light emitting meshes with *virtual\_light* on. At render time, they are stochastically sampled taking into account their intensity. The number of samples should be given as *sample* in *geometry\_light*. They can be thought of as the number of shadow rays. Note that it also affects the quality of the direct BSSRDF contribution from mesh lights.

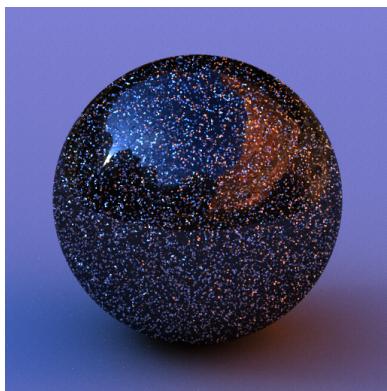
```
shader
    name "light shader example"
    virtual_light
    emission
        side "face"
        color 150 150 150
    end
end
```

## 7.3 Volumetric Properties

Here is an example of brute-force SSS shader.

```
volume
    ior          1.5
    density      0.99999 # the probability of hitting particle (per meter)
    transmittance 1 1 1
    albedo       1.0 0.5 0.0 # particle albedo
    average_cosine 0.5 # phase function
end
```

### 7.3.1 Procedural Volumetric Flakes



**redqueen** natively supports volumetric flake shader. When a ray hits an object, ray marching is performed. When a flake is found in a user specified depth, the flake overrides the color defined in the component tag. The size and density of flakes can be easily controlled. This shader consumes no memory because it is fully procedural, and exhibits no repetitive patterns.

flake		<i>unit</i>	
density	0.0	<i>scalar</i>	the density of flakes
scale	0.0	<i>m</i>	the size of each flake
depth	0.0	<i>m</i>	ray marching depth
color	0 0 0	<i>rgb</i>	albedo
end			

```

shader
    name "flake shader example"
    specular
        color 0.1 0.1 0.1
    flake
        density 0.004
        scale   0.01
        color   1 1 1
        depth   1
    end
end
normal
    color 1 1 1
    flake
        density 0.004
        scale   0.01
        color   1 1 1
        depth   1
    end
end
volume
    ior 100
end
end

```

### 7.3.2 Hair Shader



**redqueen** adopts Eugene d'Eon's hair shading model. This model is only applied for *cylinder*. **redqueen** automatically finds the root of a hair strand and obtains its UV coordinate from a mesh. Note that the hair shader interprets the parameters of the Über shader. You can assign different colors for root and tip.



```
shader
    name "hair shader example"
    glossy
        side "root"
        color 1 1 1
    end
    glossy
        side "tip"
        color 1 1 0.9
    end
    roughness
        side "both"
        color 0.2 0.5 1.0 # R, TT, TRT components
    end
    volume
        ior 1.5
    end
end
```

### References

- An energy-conserving hair reflectance model*
- Importance Sampling for Physically-Based Hair Fiber Models*
- A Practical and Controllable Hair and Fur Model for Production Path Tracing*

### 7.3.3 Random Color Shift

Random color *shift* can be used to add variation of colors based on instance/object/primitive ids.

```
shader
    name "undefined"
    emission
        side "both"
        color 0.5 0.5 0.5
        shift
            primitive_value      0.1
            primitive_hue        0.1
            primitive_saturation 0.1

            object_value         0.2
            object_hue           0.2
            object_saturation    0.2

            instance_value       0.7
            instance_hue         0.7
            instance_saturation 0.7
        end
    end
end
```

## 7.4 Volumetric Properties

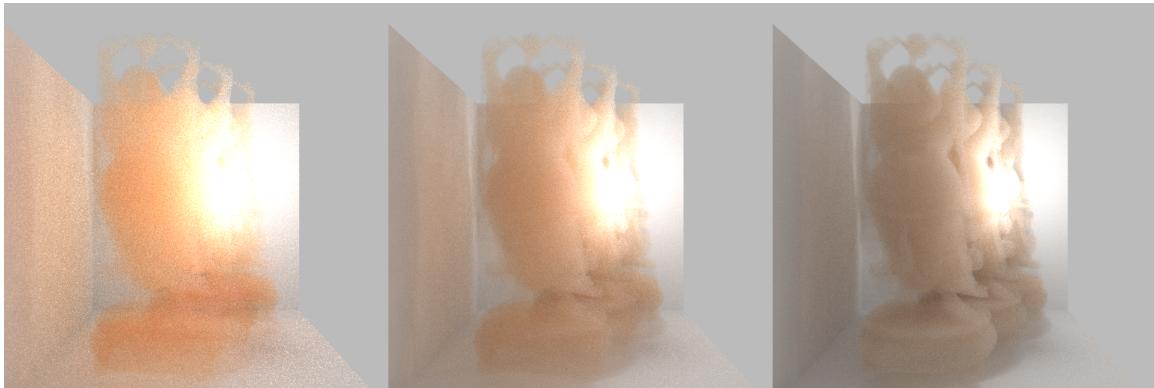


Figure 9: Rendered images with different *density* values.

Currently homogeneous media is only supported. *density* is the probability of a ray hitting particles when traveling 1 meter. *transmittance* is a light transmittance per meter. *average\_cosine* is used to determine forward/backward scattering. *albedo* can be interpreted the color of particles. *ior* is refraction index. In order to compute the illumination inside of an object, you have to set *shadow* transparency non-black.

```
shader
    name "foggy"
    specular
        side "both"
        color 1 1 1
    end
    shadow
        side "both"
        color 1 1 1
    end
    volume
        density      0.9
        average_cosine 0 # -1 to 1
        transmittance 0.8 0.3 0.1
        albedo       0.9 0.9 0.9
        ior          1
    end
end
```

## 7.5 Geometric Properties

The geometric property part is rather simple. The currently supported options are: *flip\_face*, *round\_corner*, and *smooth\_angle*. Ray traced rounded corners may be supported in the future.

```
geometry
    smooth_angle 90
    round_corner 0.1
    opacity
        color 1 1 1
        image "Model/HumanFace/shader/clip.png"
    end
    vector
        color 0 0 0.0002
        image "Model/HumanFace/shader/noise.png"
        # subdivision level
        level 2
    end
end
```

### 7.5.1 Options

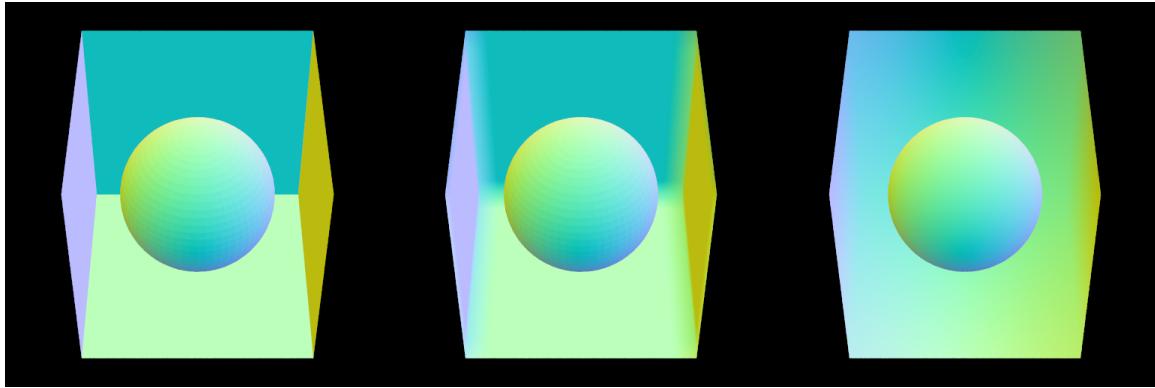


Figure 10: The *round\_corner* option can be used to round the edges of polygonal objects (middle).

### 7.5.2 Smooth Angle

Smoothed vertex normals/tangents are computed when *smooth\_angle* is greater than 0. If *smooth\_angle* is 0, geometric normals/tangents are used and user vertex normals are discarded. The default value is negative and user specified normals/tangents are used.

### 7.5.3 Component

Currently *vector* and *opacity* are supported. *opacity* only supports monochrome color and rays are stochastically terminated. *vector* can be used for vector displacement mapping. The parameter *level* is used to specify subdivision level and *vector* is a layerable component.

## 7.6 Image File Formats

**redqueen** uses FreeImage for its image I/O. Supported formats are:

```
BMP files [reading, writing]
Dr. Halo CUT files [reading]
DDS files [reading]
EXR files [reading, writing]
Raw Fax G3 files [reading]
GIF files [reading, writing]
HDR files [reading, writing]
ICO files [reading, writing]
IFF files [reading]
JBIG [reading, writing]
JNG files [reading, writing]
JPEG/JIF files [reading, writing]
JPEG-2000 File Format [reading, writing]
JPEG-2000 codestream [reading, writing]
KOALA files [reading]
Kodak PhotoCD files [reading]
MNG files [reading]
PCX files [reading]
PBM/PGM/PPM files [reading, writing]
PFM files [reading, writing]
PNG files [reading, writing]
Macintosh PICT files [reading]
Photoshop PSD files [reading]
RAW camera files [reading]
Sun RAS files [reading]
SGI files [reading]
TARGA files [reading, writing]
TIFF files [reading, writing]
WBMP files [reading, writing]
XBM files [reading]
XPM files [reading, writing]
```

More details can be found at <http://freeimage.sourceforge.net/>.

## 8 Tips for Better Quality

- For outdoor scenes, do not cast photons. Use pure path tracing. Set *resolution* 0.0 in the *render* tag.
- For indoor/semi-outdoor scenes, use final gathering. Set proper *resolution* in the *render* tag. Cast photons from each light source.
- Set *radius* 0.0 to turn off caustics photons.
- When using mesh lights, set a proper *sample* in *geometry\_light*.
- When using Image Based Lighting, set a proper *sample* in *sky\_light*.
- Basically quality is controlled by *sample* in the *render* tag. A typical setting for production quality images is 31 for *sample* of *camera* and 1024 for *sample* of *render*. Using too many anti-aliasing samples for still images is meaningless and degrades rendering performance.

## 9 Using APIs

Now you can create your own renderer plug-ins by using rq.lib. rq.lib is designed to provide extremely simple APIs for users. Please refer to rq.h for the details of APIs.

### 9.1 Rendering Flow

The flow of rendering with **redqueen** is as follows.

1. Startup (Scene independent initialization such as preparing a random number generator)
2. Create/Load Scene
3. Initialize (Scene-dependent initialization such as constructing BVHs)
4. Render
5. Finalize
6. Shutdown (Scene independent finalization)

`rqlInitialize()` does all initialization for you but you can also do step by step. In that case, please mind the order of initialization. Some shaders might need camera positions and Accelerator needs displacement mapping information. All lights needs Accelerator for photon casting. Renderer needs everything to be initialized.

1. `rqlInitializeCameras( );`
2. `rqlInitializeShaders( );`
3. `rqlInitializeAccelerator( );`
4. `rqlInitializeSkyLight( );`
5. `rqlInitializeGeometryLight( );`
6. `rqlInitializeGaffer( );`
7. `rqlInitializeRenderer( );`

### 9.2 Creating Mesh

You can create mesh objects using `rqAddObject()`. **redqueen** does not remove duplicated vertices unless you use positive *smooth\_angle* so please be aware of mesh sanity.

## Acknowledgement

I thank all beta testers, Dom Coco, Ladislav Ambruz, Makoto Tamura, Vijay Anand, and Yoichi Kimura (alphabetical order) for their awesome feedback.