

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

shinji ogaki

April 19, 2015



## 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>7</b>
5.1 Folder Structure . . . . .	7
<b>6 .rq File Format</b>	<b>7</b>
6.1 Camera . . . . .	8
6.1.1 AOVs . . . . .	9
6.2 Geometry . . . . .	10
6.2.1 Loading Geometries by .OBJ . . . . .	10
6.2.2 Loading Hair Strands by .HAIR . . . . .	10
6.2.3 Loading Geometries by Textures . . . . .	10
6.2.4 Triangle or Tetragon? . . . . .	11
6.2.5 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 Caustics . . . . .	17
6.4.6 Per Light AOVs . . . . .	17
6.4.7 Light Link . . . . .	17

<b>7</b>	<b>Shader</b>	<b>18</b>
7.1	Surface Shader . . . . .	18
7.1.1	Flags and Options . . . . .	18
7.1.2	Component . . . . .	18
7.1.3	Image File Formats . . . . .	19
7.1.4	Texture Sampler . . . . .	20
7.1.5	Roughness . . . . .	20
7.2	Surface Shader Examples . . . . .	21
7.2.1	Emitter . . . . .	21
7.2.2	Skin Shader . . . . .	22
7.2.3	Hair Shader . . . . .	24
7.2.4	Layering Surface Shader . . . . .	26
7.3	Geometry Shader . . . . .	27
7.3.1	Options . . . . .	27
7.3.2	Smooth Angle . . . . .	27
7.3.3	Component . . . . .	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
9.3	Integrating <b>redqueen</b> into Your System . . . . .	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 and small productions.
- Supports Path Tracing, Progressive Final Gathering, and Photon Mapping.
- All algorithms and shaders work with Multiple Importance Sampling.
- Every process is fully parallelized with OpenMP 2.0 and supports up to 256 threads.
- All computations are done in single precision.
- 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 shader and AOVs.

## 2 Requisites

- CPU that supports AVX instruction sets
- Please install Visual Studio 2013 Community to integrate **redqueen** into your system.
- Please install [https://software.intel.com/sites/default/files/managed/14/91/w\\_ccompxe\\_redist-msi\\_2015.0.108.zip](https://software.intel.com/sites/default/files/managed/14/91/w_ccompxe_redist-msi_2015.0.108.zip) as **redqueen** switched to Intel Compiler.

## 3 TODO List

- Culling shadow rays in Multiple Importance Sampling
- Lens Shader (including Flare)
- Data Compression with FastPFor
- Better Multiple Importance Sampling for Textured Triangular Meshes
- Better Intersection Test for Hair Primitives
- Better Photon Mapping
- Better Memory Layout for Faster Rendering

## 4 Change Log

✗ incompatible + new feature ↗ improvement • bug fix

2015		Class	Details
19 Apr	✗	API	New simplified APIs

2015		Class	Details
04 Jan	+ +	<i>object</i> <i>sampler</i>	<i>triangle_texture</i> is added. Texture sampler supports stochastic bilinear filter.
03 Jan	+ ✗ ✗	<i>object</i> <i>API</i> <i>cylinder</i>	<i>instance</i> is added to support multi-level instancing. new APIs to create objects <i>assign_hair_texture_uv</i> is no longer needed.
01 Jan	+ • • •	<i>tree</i> <i>geometry_light</i> <i>parallel_light</i> <i>object</i>	Multi-level instancing with motion blur Minor bug in sampling Minor bug in photon casting Robust planar check of tetragons

2014		Class	Details
01 Nov	●	tree	Robust traversal
31 Oct	↗ ↗ ↗	sky_light render object	Less memory consumption Each photon consumes less memory. .OBJ loader is 30% faster.
26 Oct	↗ ● ● ● ●	redqueen render render render	Switched to Intel Compiler, which gives 5% speed up. Redundant photon mapping computation Uninitialized value caused freezing. Vector displacement was broken.
22 Oct	● ●	AOVs redqueen	Images were not saved properly. Hash function memory problem
04 Aug	✗ ✗	geometry_shader geometry_shader	Visibility flags such as <i>invisible</i> are moved to <i>geometry_shader</i> <i>clip</i> is moved to <i>geometry_shader</i>
01 Aug	✗ + + + ↗ ↗ ↗ ↗ ● ● ● ● ● ● ● ●	object group light geometry_shader AOV object geometry_shader parallel_light object object tree hair_shader geometry_light	<i>ignore_vertex_vector</i> and <i>smooth_vertex_vector</i> are no longer supported. Arbitrary per vertex data is supported. Link between <i>light</i> and <i>shader</i> <i>smooth_angle</i> is supported. Arbitrary per vertex data is supported. More useful stats Smoothing vertex vectors is fully parallelized. Displacement mapping is fully parallelized. Direction is flipped when using blur. Better hash function for faster initialization Broken normal/tangent when using instance Missing tetragon on rare occasions Small <i>roughness</i> causes crash. MIS weight computation
14 Mar	✗ + + +	camera shader API cylinder	<i>exposure</i> is relocated to the camera tag. Eugene's hair shader Functions to write AOVs Automatic hair UV assignment
16 Feb	+ ↗ ↗ ↗	AOV tree tree shader	Per light AOVs (Multilight) SATO (Surface-area traversal order) gives 5% speedup for shadow rays. Faster construction on multi socket systems Texture sampler uses a stochastic bilinear filter.
10 Feb	↗ ↗ ↗	shader render tree	SSS is 5-50% faster. Density estimation is done with pre-multiplied color. SAH Computation is improved. This requires slightly more memory.
11 Jan	↗ ↗ ↗ ● ●	tree object object sky_light render	10-25% speedup Packed vectors now use 24bits for faster decoding. Negative ids are supported in OBJ. Redundant computation in MIS Poor performance on multi socket systems

2013		Class	Details
15 Dec	↗ ● ● ●	<i>object</i> <i>geometry_light</i> <i>object</i> <i>shader</i>	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	↗ ● ● ● ●	<i>object</i> <i>geometry_shader</i> <i>tree</i> <i>object</i> <i>object</i>	.OBJ loader is 30% faster. <i>smooth_vertex_vector</i> did not work for hair strands (cylinders) Missing objects when rendering gigantic data sets Packed vectors now use 32bits not 16bits. This slows down a bit. Degenerated tetragon removal
07 Dec	↗ ↗	<i>photon_map</i> <i>photon_map</i>	nearly 200% faster photon gathering for a large search radius photon mapping uses the <i>offset</i> option of non-area lights
05 Dec	✚ ↗ ●	<i>API</i> <i>API</i> <i>photon_map</i>	Functions to load multiple particles/cylinders/triangles/tetragons at once Switched from static lib to dynamic lib. Dark caustics issue
31 Nov	✗ ✚ ●	<i>render</i> <i>object</i> <i>shader</i>	Using negative values skips clamping. Convex planar quadrilateral is supported as an atomic primitive. High glossy materials render black.
25 Nov	↗ ↗ ↗ ●	<i>tree</i> <i>tree</i> <i>tree</i> <i>object</i>	Tree construction is 10% faster. Tree node uses 2/3 memory. Faster geometry loading Degenerated triangle removal
22 Nov	✚ ↗	<i>redqueen</i> <i>tree</i>	Created Visual Studio 2013 version, which is slightly faster. Peak memory consumption during tree construction is greatly reduced.
14 Nov	✗ ✚ ✚ ✚ ✚ ✚ ✚ ●	<i>shader</i> <i>shader</i> <i>point_light</i> <i>point_light</i> <i>parallel_light</i> <i>object</i> <i>object</i> <i>object</i>	<i>angle map</i> uses $[0, 1]$ normalized value (mapped to $[0, 2\pi]$ ) not degree. Layerable <i>surface_shader</i> with <i>weight</i> control Spot light <i>offset</i> is added to prevent jagged shadows. <i>offset</i> 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	✗ ✚ ✚	<i>object</i> <i>object</i> <i>redqueen</i>	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	↗	<i>redqueen</i>	3% speed up
28 Oct	✚ ✚	<i>camera</i> <i>object</i>	Render region Faster mesh loading via textures
26 Oct	● ● ●	<i>sky_light</i> <i>redqueen</i> <i>documentation</i>	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	✗ ✚ ✚ ● ●	<i>display</i> <i>shader</i> <i>object</i> <i>shader</i> <i>triangle</i>	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	✚ ↗ ● ●	<i>object</i> <i>tree</i> <i>shader</i> <i>geometry_light</i>	Particle is now activated. Faster .OBJ loader (50% speedup for single thread and more for multi-thread) Smoothing normal issue Sampling issue
10 Oct	✚ ↗ ● ●	<i>shader</i> <i>tree</i> <i>shader</i> <i>geometry_light</i>	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 <i>geometry_light</i>
06 Oct	●	<i>shader</i>	Density estimator and MIS

## 5 Loading Your Scene

### 5.1 Folder Structure

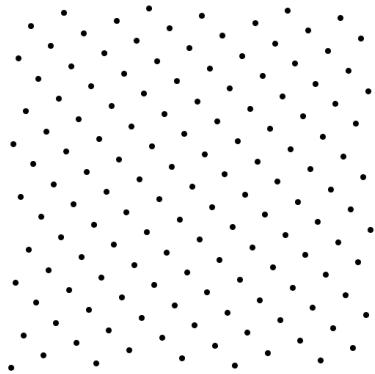
**redqueen** takes the name of folder that contains your scene as its argument. The folder should be organized as follows for easy asset management.

```
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*, *transmissive*, *glossy*, *specular*, *emissive*, *normal*, *tangent*, *depth*, and *position*. Users can write per vertex data into an image.

```
rqAddVertexData ( object_id, group_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 *group* 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

### 6.2.1 Loading Geometries by .OBJ

Exporting one gigantic OBJ file is not recommended. Using many small OBJ files leads to less memory consumption.

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

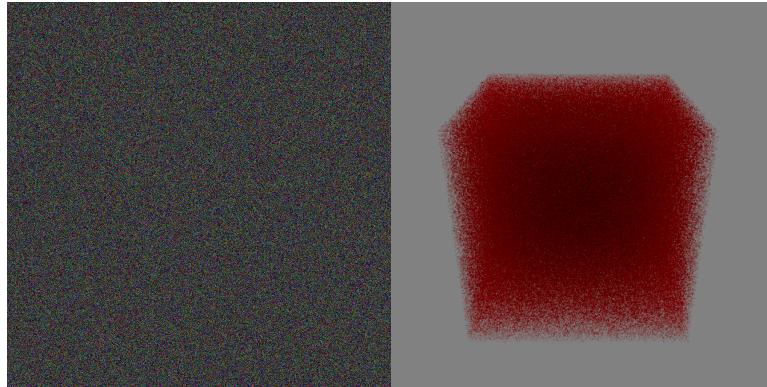
### 6.2.2 Loading Hair Strands by .HAIR

**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.

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

The HAIR models are available at <http://www.cemyuksel.com/research/hairmodels/>.

### 6.2.3 Loading Geometries by Textures



Users can load data by binary formats, more specifically via textures. This is at least an order of magnitude faster than loading scenes via OBJ. **This provides better readability and simple data compression through codec.**

The simplest example is loading particles. The left image is contains 1 million particle data. The positions are described in the RGB channel and radii are in alpha. By using an EXR file, 1 million particles can be loaded in a second.

<b>object</b>			
name	"set1"	<i>unit</i>	object name
shader	"red_particles"	<i>string</i>	the name of a shader to assign to particles
particle_texture	"particle_set1.exr"	<i>string</i>	the name of texture that stores particles
<b>end</b>			

Mesh data can also be loaded by using textures. This realizes the fastest mesh loading. *position\_texture*, *normal\_texture*, *tangent\_texture*, and *uv\_texture* can have a different size. These 4 textures must have RGB channels, for XYZ or UVW. have the same number of pixels, which is identical to the number of triangles. These

4 textures must have RGB channels, for 3 triangle vertices. All textures for storing triangle data must be in 32bit float (single precision floating point) format. You cannot write multiple same textures in one *object*. And please use lossless compression so that your scene can be properly created.

<b>object</b>				
name	"mesh1"	<i>string</i>	unit	object name
shader	"plastic"	<i>string</i>		shader to assign
position_texture	"position.pfm"	<i>string</i>		texture that stores vertex positions
normal_texture	"normal.pfm"	<i>string</i>		texture that stores vertex normals
tangent_texture	"tangent.pfm"	<i>string</i>		texture that stores vertex tangents
uv_texture	"uv.pfm"	<i>string</i>		texture that stores vertex uvs
triangle_texture	"triangle.pfm"	<i>string</i>		texture that stores vertex position ids
<b>end</b>				

#### 6.2.4 Triangle or Tetragon?

**redqueen** supports planar convex quadrilaterals. They are very common in architectural scenes. Many other renderers only support triangular meshes. On the other hand, **redqueen** adopts quadrilateral as one of atomic primitives since it provides a couple of merits: 1) compact topological data of meshes, 2) a shallower tree due to less number of primitives, 3) faster tree construction.

#### 6.2.5 Multi-Level Instancing



Figure 1: Multi-level instancing. Photon casting works fine for instanced light emissive objects.

<b>instance</b>				
object	"name"	<i>string</i>	unit	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
<b>end</b>				

**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.

```

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

object
    name "scene"
    obj  "model\Instance\geometry\floor.obj"
    instance
        matrix 1 0 0 -2      0 1 0 0      0 0 1 -2      0 0 0 1
        object "teapot"
    end
    instance
        matrix 1 0 0 2      0 1 0 0      0 0 1 -2      0 0 0 1
        object "teapot"
    end
    instance
        matrix 1 0 0 2      0 1 0 0      0 0 1 2      0 0 0 1
        object "teapot"
    end
    instance
        matrix 1 0 0 -2      0 1 0 0      0 0 1 2      0 0 0 1
        object "teapot"
    end
end

object
    name "teapot"
    obj  "model\Instance\geometry\teapot.obj"
    instance
        matrix 1 0 0 -1      0 1 0 2      0 0 1 -1      0 0 0 1
        object "sphere"
    end
    instance
        matrix 1 0 0 -1      0 1 0 2      0 0 1 1      0 0 0 1
        object "sphere"
    end
    instance
        matrix 1 0 0 1      0 1 0 2      0 0 1 -1      0 0 0 1
        object "sphere"
    end
    instance
        matrix 1 0 0 1      0 1 0 2      0 0 1 1      0 0 0 1
        object "sphere"
    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*

### 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>	smart clamping to get rid of fire flies
gamma	2.2	<i>scalar</i>	display gamma
<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>
distance	0.2	<i>m</i>	distance for bias compensation/secondary FG a.k.a. retrace threshold
radius	0.1	<i>m</i>	photon search radius
resolution	0.05	<i>m</i> <sup>2</sup>	density estimation resolution
end			
end			

### 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	"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>			

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	"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>			

## 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 *emissive* 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>	"emitter1"	<i>string</i>	the name of a shader with an emissive component
<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	"transmissive.png"	<i>string</i>	the name of aov image file
<b>end</b>			
<b>aov</b>			options for per light AOVs
<b>shader</b>	"emitter2"	<i>string</i>	the name of a shader with an emissive component
<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	"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*
- Fast Random Sampling of Triangular Meshes*

#### 6.4.4 Sky Light

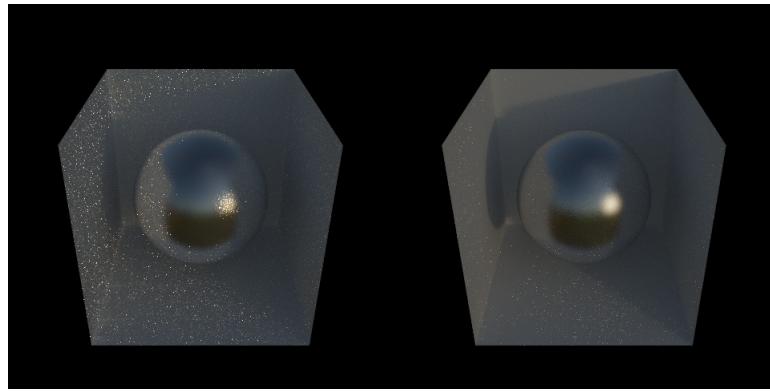


Figure 3: *sky\_light* MIS. The left image was rendered without MIS and is missing the illumination from the sun. The average pixel values are 0.496 for the left image and 0.497 for the right image, respectively.

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 Caustics

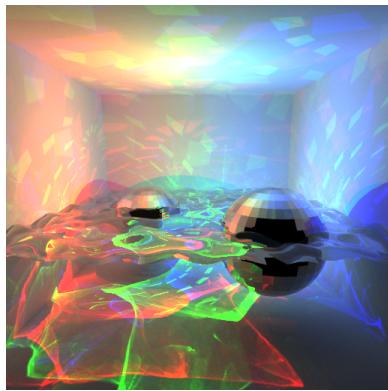


Figure 4: Caustics

Caustics can be rendered by casting photons from each light. Caustics by geometry light can be rendered by path tracing.

#### 6.4.6 Per Light AOVs



Figure 5: Per Light AOVs

**redqueen**'s per light AOVs work in the same way as the Multilight (<http://support.nextlimit.com/display/mxdocsv3/Multilight>). Here is an example showing what you can do with per light AOVs. By using the per light AOVs, you can modify the lighting effect afterwards. You can also tweak the lighting per component such as lambertian, glossy, and specular.

#### 6.4.7 Light Link

## 7 Shader

This section describes how to use **redqueen**'s Über shader. This is the most complicated part of **redqueen**. The Über shader consists of 3 shaders: 1. *surface\_shader*, 2. *geometry\_shader*, and 3. *volume\_shader*. **Currently, *volume\_shader* is ignored.**

<b>shader</b>	<i>unit</i>
<b>surface_shader</b>	
<b>end</b>	
<b>geometry_shader</b>	
<b>end</b>	
<b>volume_shader</b>	
<b>end</b>	
<b>end</b>	

### 7.1 Surface Shader

#### 7.1.1 Flags and Options

The currently supported flags are: *window*, *thin\_dielectric*, *dielectric*, *camera\_primary*, *light\_primary*, *virtual\_light*, and *invisible*. The sideness options are: "face", "back", and "both".

#### 7.1.2 Component

**redqueen** supports the following basic color components: *lambertian*, *transmissive*, *glossy*, *specular*, and *emissive*. Each basic component works with these components: *diffusion*, *ior*, *roughness*, *transparency*, *angle*, *anisotropy*, *clip*, *weight*, and *normal*. All components have the same parameters so once you understand how to use it, you can create any kind of materials.

<i>name_of_component</i>		<i>unit</i>	
color	1 1 1	<i>rgb</i>	the type of component
image	"texture.png"	<i>string</i>	color or multiplier for texture
level	1	<i>integer</i>	the name of texture map
sampler			not used in <i>surface_shaders</i> but in <i>geometry_shaders</i>
end			texture sampler
end			

#### References

*Bounding the Albedo of the Ward Reflectance Model*

### 7.1.3 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/>.

#### 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.

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

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

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

In **redqueen**,

$$(U', V', W' = 1)^T \neq \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} (U, V, W = 1)^T. \quad (6)$$

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

$$R' = (aR + bG + cB + dE)/(mR + nG + oB + pE) \quad (7)$$

$$G' = (eR + fG + gB + hE)/(mR + nG + oB + pE) \quad (8)$$

$$B' = (iR + jG + kB + lE)/(mR + nG + oB + pE) \quad (9)$$

$$E' = (mR + nG + oB + pE)/(mR + nG + oB + pE). \quad (10)$$

In **redqueen**,

$$(R', G', B', E' = 1)^T \neq \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} (R, G, B, E = 1)^T. \quad (11)$$

#### 7.1.5 Roughness

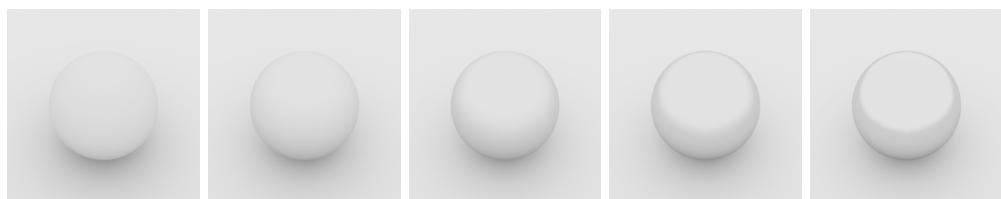


Figure 6: Roughness. From left to right: 1, 0.5, 0.25, 0.125, and 0.0625

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

## 7.2 Surface Shader Examples

### 7.2.1 Emitter

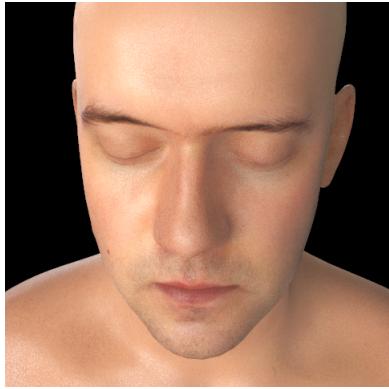


Figure 7: *geometry\_light* MIS. The left image does not use *virtual\_light*. The center image was rendered *virtual\_light* on. Using clamp generates clean image (right).

**redqueen** automatically generates many virtual light sources on light emitting meshes that have *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"
    surface_shader
        virtual_light 1
        sideness      "face"
        emissive
            color 10 10 10
            image "image.exr"
            sampler
                gamma 2.2
            end
        end
    end
    surface_shader
        virtual_light 0
        sideness      "back"
        emissive
            color 0 0 0
        end
    end
end
```

## 7.2.2 Skin Shader



Originally Lambertian/Diffuse models is developed to approximate Sub Surface Scattering. Ray-Traced SSS works when *lambertian* and *diffusion* are properly set. *lambertian* is used as surface color and *diffusion* is used as the variance of a gaussian profile. If the color of *diffusion* is zero, your material is rendered as a normal lambertian surface. Too large variance blurs out the details of shading. This SSS is realized by simply jittering shading points, and color is compensated by a user specified profile. The quality of direct BSSRDF is controlled by *sample* in lights and the indirect BSSRDF by *sample* in the *render* tag. A must read material can be found here: <http://www.iryoku.com/projects/nextgen/downloads/Next-Generation-Character-Rendering-v6.pptx>. And an article from GPU Gems3 gives you the idea of how to choose diffusion parameters. [http://http.developer.nvidia.com/GPUGems3/gpugems3\\_ch14.html](http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html). They should be specified in [m] in **redqueen**. Here is an example of skin shader.

```

shader
    name "Face"
    surface_shader
        # skin color
        lambertian
            color 1 1 1
            image "Model/HumanFace/shader/lambertian.jpg"
            sampler
                # correct gamma
                gamma 2.2
            end
        end
        # add micro scale roughness
        normal
            color 1 1 1
            image "Model/HumanFace/shader/normal.png"
            sampler
                # tile texture 4 x 4
                uv_matrix 4 0 0    0 4 0    0 0 1
            end
        end
        diffusion
            color 0.00036 0.0002 0.0001
        end
        # simulation of pore
        glossy
            color 0.8 0.8 0.8
            image "Model/HumanFace/shader/pore.jpg"
            sampler
                uv_matrix 32 0 0    0 32 0    0 0 1
                gamma 2.2
            end
        end
        roughness
            color 0.18 0.18 0.18
        end
        # refractive index of water
        ior
    
```

```
        color 1.33 1.33 1.33
    end
end
geometry_shader
    smooth_angle 90
    # add large scale bumpiness
    vector
        color 0 0 0.0002
        image "Model/HumanFace/shader/noise.png"
        # subdivision level
        level 2
    end
end
end
```

## References

- Bidirectional Lightcuts (Appendix B)*
- BSSRDF Importance Sampling*

### 7.2.3 Hair Shader



Figure 8: Eugene d'Eon's energy conserving model and automatic hair UVW assignment



Figure 9: Eugene d'Eon's energy conserving model and automatic hair UVW assignment

**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. Layering *surface\_shader* is not supported.

```
shader
    name "hair"
    surface_shader
        dielectric 1
        glossy
            color 0.8 0.8 0.8
            image "model\fur\shader\checker.jpg"
        end
        ior
            color 1.5 1.5 1.5
        end
        roughness
            color 0.15 0.15 0.15
        end
    end
```

```
geometry_shader
    smooth_angle          90
end
volume_shader
    # transmittance per hair diameter
    transmittance 0.99 0.99 0.99
end
end
```

## References

- An energy-conserving hair reflectance model*  
*Importance Sampling for Physically-Based Hair Fiber Models*

#### 7.2.4 Layering Surface Shader

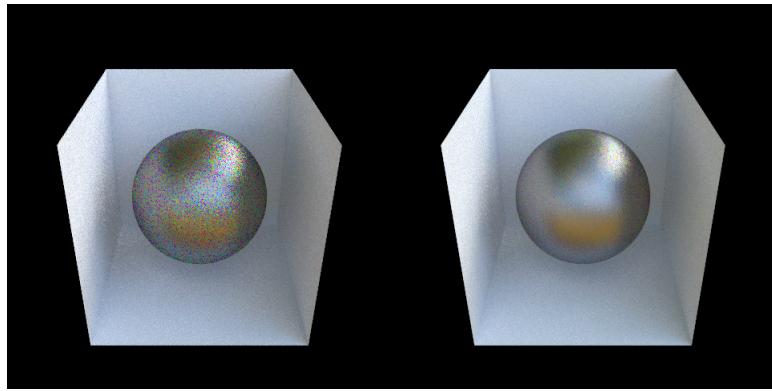


Figure 10: Layering R, G, and B colored glossy components. The left image uses *sample* = 4 and right uses *sample* = 31.

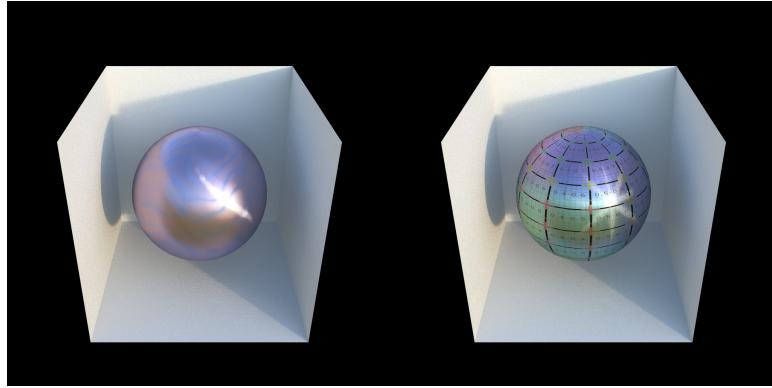


Figure 11: Complicated highlight patterns with *weight* maps.

**redqueen** supports one shader per primitive but *surface\_shader* is layerable. In other words, *shader* can have multiple *surface\_shaders*. The maximum number of *surface\_shaders* is limited to 256. If surface shaders have different colors, rendered images end up having chromatic noise. In order to reduce it, you need a relatively large value for *sample* in *camera*. Layering *surface\_shader* takes a little bit of time so should not be used when it is unnecessary. You might be wondering why you cannot use multiple components such as *glossy* and *diffusion* instead of multiple *surface\_shaders*. This is because each component can have different *normal* and *ior* maps so that more flexible control is possible.

## 7.3 Geometry Shader

*geometry\_shader* is rather simple and very similar to *surface\_shader*.

### 7.3.1 Options

The currently supported options are: *flip\_face* and *smooth\_angle*. The *round\_corner* option can be used to round the edges of polygonal objects. *reduction* and *trivial\_connection* will be supported in the future.

### 7.3.2 Smooth Angle

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

#### Why smoothing normals/tangents is slow?

**redqueen** automatically removes duplicated vertices, which takes some time. Although the procedure is parallelized, memory traffic has a great impact. Duplicated points are also removed when converting legacy *object* to **redqueen**'s native format.

### 7.3.3 Component

Currently *vector* is only supported. *vector* can be used for vector displacement mapping. The parameter *level* is used to specify subdivision level and *vector* is a layerable component.

## 8 Tips for Better Quality

- For outdoor scenes, do not cast photons. Use pure path tracing. Set *distance* and *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*.
- If your monitor is properly calibrated, set 1.0 for *gamma* in the *render* tag.
- 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 (preparation of random number generator, etc.)
2. Create/Load Scene
3. Initialize (preparation of importance samplers, acceleration structure construction, casting photons, etc.)
4. Render
5. Finalize
6. Shutdown

`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. `rqlInitializeCameraLibrary( );`
2. `rqlInitializeShaderLibrary( );`
3. `rqlInitializeAccelerator( );`
4. `rqlInitializeSkyLight( );`
5. `rqlInitializeGeometryLight( );`
6. `rqlInitializeGaffer( );`
7. `rqlInitializeRenderer( );`

### 9.2 Creating Mesh

You can create mesh objects in two ways. One is using `rqAddObject()` and the other is through `rqAddGroup()`. **group** is a new data structure implemented to support arbitrary vertex data. Each *object* is converted to *group*. Therefore, there is an overhead using *object*. If you create meshes by `rqAddGroup()`, **redqueen** does not remove duplicated vertices unless you use positive *smooth\_angle*. You can get dramatic speed up but please be aware of mesh sanity.

Displacement mapping and smoothing vertex normals/tangents are performed per *group*. To make smoothing normals/tangents and displacement mapping work, *particle*, *cylinder*, and polygons (*triangle* and *tetragon*) should be in different groups.

As mentioned earlier, each *group* has several vertex data as default: position, reference (pref), motion, uv, normal, tangent, and radius. These are very common vertex data.

### 9.3 Integrating redqueen into Your System

When creating a project with Visual Studio, add rq.lib and gdiplus.lib, and select /MT for Runtime Library. Note also that **redqueen** only supports x64. Please refer <https://software.intel.com>.

## Acknowledgement

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