redqueen user's manual - simple work flow for everyone

shinji ogaki

July 1, 2015

Contents

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 OBVH and uses 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 shaders 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` as **redqueen** switched to Intel Compiler.

3 TODO List

- Python Binding
- T-SAH BVH Builder
- Reduce Occlusion Tests in Multiple Importance Sampling
- Mesh Processing
- Faster Intersection Test for Hair Primitives
- Better Memory Layout for Faster Rendering
- Better Multiple Importance Sampling for Textured Triangular Meshes
- Microflake (SGGX)
- Switch from OpenMP to C++11 Threading
- Remove All Critical Sections and Atomic Operations
- Sharing Data with User Applications
- Lens Shader (including Flare)

4 Change Log

≠ incompatible + new feature ↗ improvement • bug fix

2015		Class	Details
30 Jun	↗	*shader*	Map for displacement is loaded on demand and discarded immediately after use.
	↗	*redqueen*	Better task scheduling leads to less working memory
	↗	*sky_light*	Stochastic bilinear filter is used by default to speed up texture mapping.
	↗	*tree*	If the number of primitives is over 16.7million, **redqueen** switches to memory save mode.
27 Jun	↗	*cylinder*	Automatic hair uv assignment is parallelized.
	↗	*tree*	BVH for static objects uses far less memory.
18 Jun	↗	*tree*	Reduced the size of pre-allocated memory for BVH
17 Jun	≠	*renderer*	*distance* parameter for secondary final gathering is removed. Now it's calculated automatically.
	↗	*geometry*	Working memory is reduced when calculating smoothed vertex vectors.
16 Jun	↗	*tree*	Consumes 4 bytes less memory per primitive
12 Jun	↗	*sky_light*	Smaller LUT for importance sampling
	●	*renderer*	Trace depth
	●	*sky_light*	AOV channel check
11 Jun	●	*shader*	Artifacts caused by uninitialized uv coord
10 Jun	↗	*tree*	Improved build performance on multi-socket systems
	↗	*object*	Improved hair intersection test
	↗	*object*	AVX2 is not required for hair intersection test.
09 Jun	↗	*photon_map*	Search radius is adaptively changed based on the number of bounces.
	↗	*photon_map*	Each photon consumes 4 bytes less memory.
	↗	*object*	.OBJ loader uses less memory.
	●	*object*	Artifacts of highly tessellated meshes
	●	*shader*	Very low roughness value caused numerical error.
02 Jun	↗	*object*	Each triangle/tetragon consume 4 bytes less memory.
11 May	↗	*geometry_light*	Increased accuracy in MIS computation
	↗	*sky_light*	Increased accuracy in MIS computation
	●	*API*	Display gamma was broken
09 May	+	*tree*	Improved version of "Child Node Sorting for Fast Occlusion Test"
	↗	*tree*	A slightly smaller memory footprint
	↗	*tree*	Faster occlusion test
	●	*tree*	Robust intersection test for multi-level instancing
	●	*tree*	Crash when instanced object refers to non-existing object
	●	*tree*	Crash when empty object is used
	●	*AOV*	Camera AOVs are broken due to the change of APIs.
04 May	●	*shader*	Fixed fresnel term for *thin_dielectric*
	●	*tree*	Occasional crash due to insufficient stack size
02 May	≠	*shader*	Clip map is removed and opacity map is implemented.
30 Apr	+	*renderer*	Color of error pixels (black by default)
	↗	*sky_light*	Improved stratification and initialization
26 Apr	↗	*image*	Date and time is added to the file name of beauty.
	●	*sky_light*	Better stratification
	●	*shader*	Smoothing normals
21 Apr	●	*shader*	Minor bug in shadow transparency
20 Apr	+	*object*	Runtime tessellation of hair
19 Apr	≠	*API*	New simplified APIs
04 Jan	+	*object*	*triangle_texture* is added.
	+	*sampler*	Stochastic bilinear filter for image sampling
03 Jan	+	*object*	*instance* is added to support multi-level instancing.
	≠	*API*	new APIs to create objects
	≠	*cylinder*	*assign_hair_texture_uv* is no longer needed.
01 Jan	+	*tree*	Multi-level instancing with motion blur
	●	*geometry_light*	Minor bug in sampling
	●	*parallel_light*	Minor bug in photon casting
	●	*object*	Robust planar check for tetragons

2014		Class	Details
01 Nov	•	*tree*	Robust traversal
31 Oct	↗	*sky_light*	Less memory consumpsion
	↗	*render*	Each photon consumes less memory.
	↗	*object*	.OBJ loader is 30% faster.
26 Oct	↗	*redqueen*	Switched to Intel Compiler, which gives 5% speed up.
	•	*render*	Redundant photon mapping computation
	•	*render*	Uninitialized value caused freezing.
	•	*render*	Vector displacement was broken.
22 Oct	•	*AOVs*	Images were not saved properly.
	•	*redqueen*	Hash function memory problem
04 Aug	≠	*geometry_shader*	Visibility flags such as *invisible* are moved to *geometry_shader*
	≠	*geometry_shader*	*clip* is moved to *geometry_shader*
01 Aug	+	*part*	Arbitrary per vertex data is supported.
	+	*light*	Link between *light* and *shader*
	+	*geometry_shader*	*smooth_angle* is supported.
	↗	*AOV*	Arbitrary per vertex data is supported.
	↗	*object*	More useful stats
	↗	*geometry_shader*	Smoothing vertex vectors is fully parallelized.
	↗	*geometry_shader*	Displacement mapping is fully parallelized.
	•	*parallel_light*	Direction is flipped when using blur.
	•	*object*	Better hash function for faster initialization
	•	*object*	Broken normal/tangent when using instance
	•	*tree*	Missing tetragon on rare occasions
	•	*hair_shader*	Small *roughness* causes crash.
	•	*geometry_light*	MIS weight computation
14 Mar	≠	*camera*	*exposure* is relocated to the camera tag.
	+	*shader*	Eugene's hair shader
	+	*API*	Functions to write AOVs
	+	*object*	Automatic hair UV assignment
16 Feb	+	*AOV*	Per light AOVs (Multilight)
	↗	*tree*	SATO (Surface-area traversal order) gives 5% speedup for shadow rays.
	↗	*tree*	Faster construction on multi-socket systems
	↗	*shader*	Texture sampler uses a stochastic bilinear filter.
10 Feb	↗	*shader*	SSS is 5-50% faster.
	↗	*render*	Density estimation is done with pre-multiplied color.
	↗	*tree*	SAH Computation is improved. This requires slightly more memory.
11 Jan	↗	*tree*	10-25% speedup
	↗	*object*	Packed vectors now use 24bits for faster decoding.
	↗	*object*	Negative ids are supported in OBJ.
	•	*sky_light*	Redundant computation in MIS
	•	*render*	Poor performance on multi-socket systems

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

5 Loading Your Scene

5.1 Folder Structure

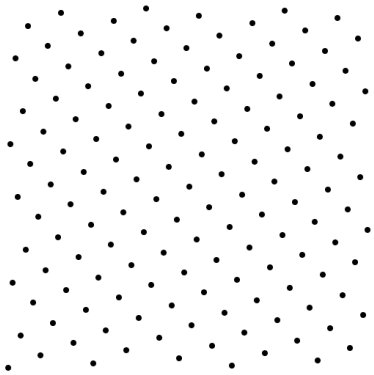
redqueen takes the name of folder that contains your scene as its argument. The folder should be organized as follows for easy asset management.

```
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 1×1, 2×2, 3×3... 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).

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

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, 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

6.2.1 Loading Geometry 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)
```

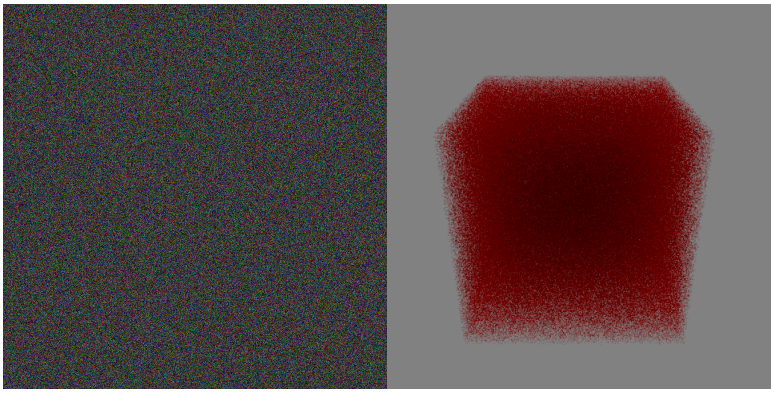
6.2.2 Loading Hair Strands by .HAIR

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.

object		*unit*	
name	"set1"	*string*	object name
shader	"red_particles"	*string*	the name of a shader to assign to particles
particle_texture	"particle_set1.exr"	*string*	the name of texture that stores particles
end			

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.

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

6.2.4 Triangle or Tetragon?

redqueen supports regular tetragons. 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

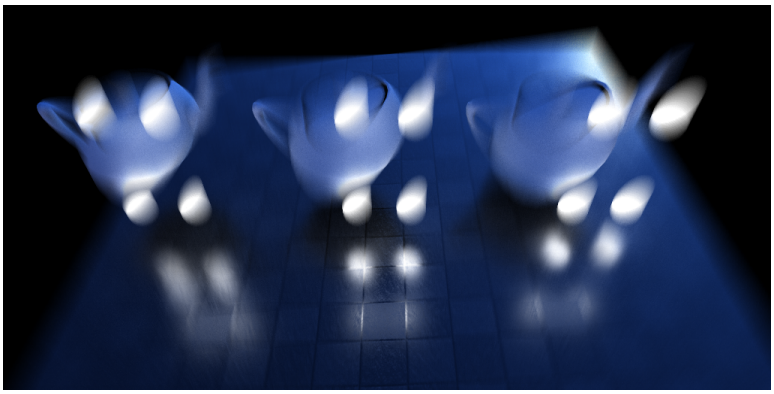


Figure 1: Multi-level instancing. Photon casting works fine for instanced light emissive objects.

instance		*unit*	
object	"name"	*string*	object name
shader	"plastic"	*string*	shader to assign
matrix	1 0 0 0 0 1 0 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
end			

redqueen supports multi-level instancing (up to 16 levels). Every *object* can have instanced *object*s 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
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		*unit*	
clamp	1 1 1	*rgb*	clamping to get rid of fire flies
gamma	2.2	*scalar*	display gamma
error	100 0 0	*rgb*	color for sub sample that has numerical error
image_path	"../images"	*string*	image file search path
image_path	"../../hdr"	*string*	image file search path
final_gather			options for the FG integrator
sample	256	*integer*	indirect GI samples for both *lambertian* and *glossy*
ray_depth	8	*integer*	ray trace depth (eye path)
photon_depth	8	*integer*	photon trace depth (light path)
bounce	2	*integer*	GI bounce both for *lambertian* and *glossy*
radius	0.1	m	photon search radius
resolution	0.05	m^2	density estimation resolution
end			
end			

Figure 2: 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

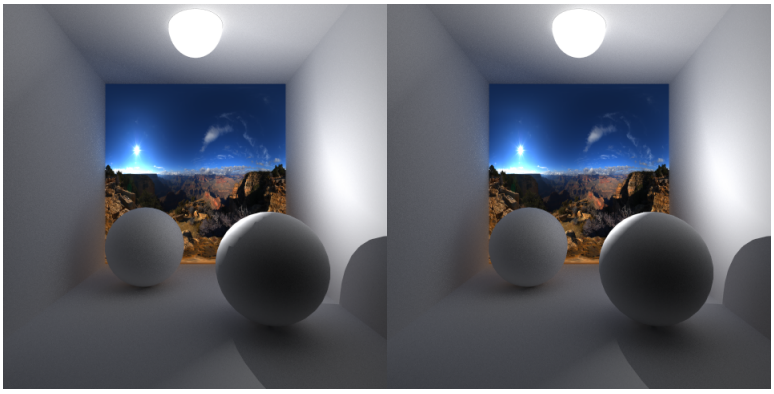


Figure 3: By setting *offset* 0.1, the jagged shadow problem on the sphere is solved.

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

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

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

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

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

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

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.

geometry_light		*unit*	
photon	100000	*integer*	the number of photons to shoot
sample	128	*integer*	the number of samples for shadow rays and direct BSSRDF
aov			options for per light AOVs
shader	"emitter1"	*string*	the name of a shader with an emissive component
type	"specular"	*string*	component you want write out
type	"lambertian"	*string*	component you want write out
type	"glossy"	*string*	component you want write out
image	"transmissive.png"	*string*	the name of aov image file
end			
aov			options for per light AOVs
shader	"emitter2"	*string*	the name of a shader with an emissive component
type	"specular"	*string*	component to write out
type	"lambertian"	*string*	component to write out
type	"glossy"	*string*	component to write out
image	"transmissive.png"	*string*	the name of aov image file
end			
link			options for light link
shader	"shader_name"	*string*	the name of shader
end			
end			

References

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

6.4.4 Sky Light

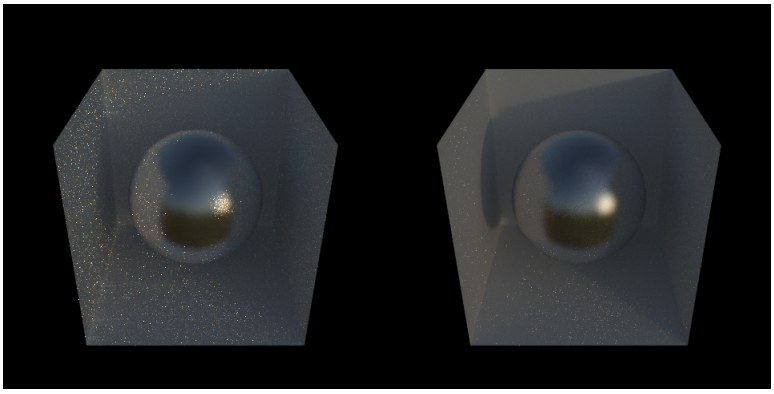


Figure 4: *sky_light* MIS. The left image was rendered without MIS and is missing the illumination from the sun. The average pixel values are 0.496 for the left image and 0.497 for the right image, respectively.

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

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

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

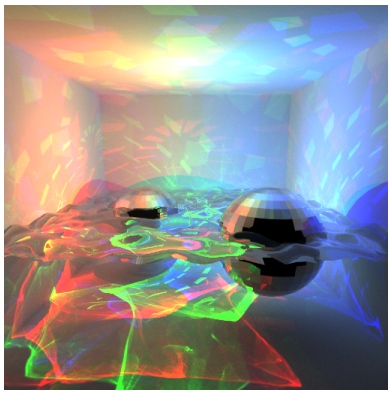


Figure 5: Caustics

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

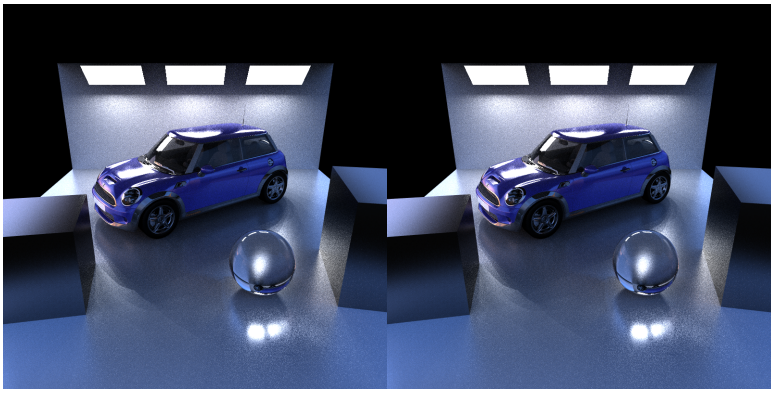


Figure 6: Per Light AOVs

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.

shader	*unit*
surface_shader	
end	
geometry_shader	
end	
volume_shader	
end	
end	

7.1 Surface Shader

7.1.1 Flags and Options

The currently supported flags are: *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*, *angle*, *anisotropy*, *weight*, and *normal*. All components have the same parameters so once you understand how to use it, you can create any kind of materials.

name_of_component		*unit*	the type of component
color	1 1 1	*rgb*	color or multiplier for texture
image	"texture.png"	*string*	the name of texture map
level	1	*integer*	not used in *surface_shader*s but in *geometry_shader*s
sampler			texture sampler
end			
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		*unit*	
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 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

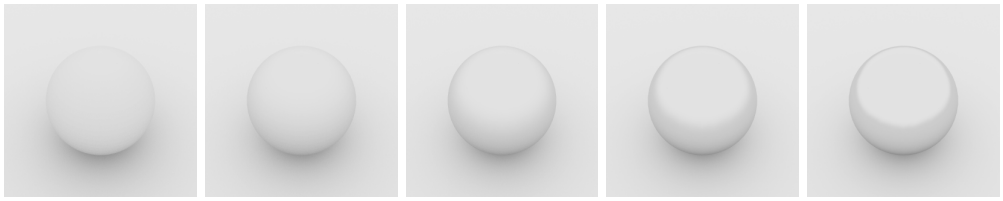


Figure 7: Roughness. From left to right: 1, 0.5, 0.25, 0.125, and 0.0625

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

7.2 Surface Shader Examples

7.2.1 Emitter

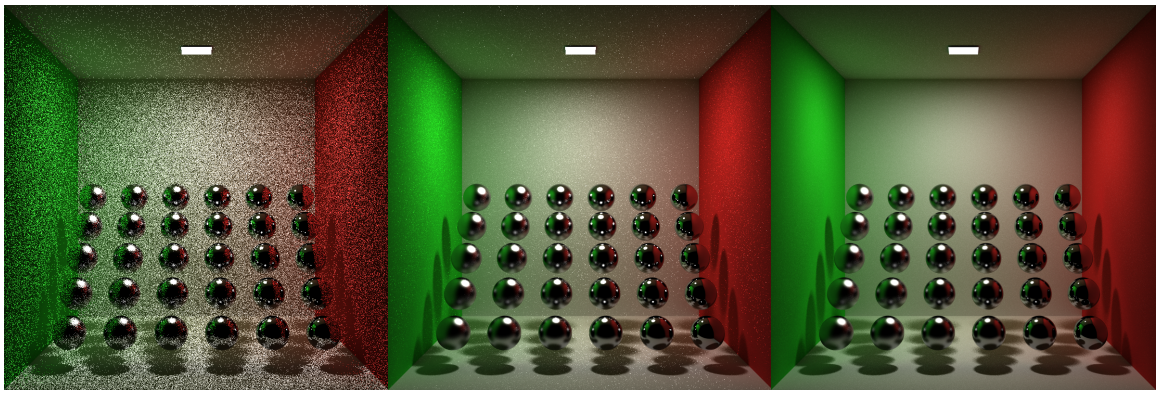


Figure 8: *geometry_light* MIS. The left image does not use *virtual_light*. The center image was rendered *virtual_light* on. Using clamp generates clean image (right).

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
        sideness      "face"
        emissive
            color 10 10 10
            image "image.exr"
            sampler
                gamma 2.2
            end
        end
    end
    surface_shader
        sideness      "back"
        emissive
            color 0 0 0
        end
    end
end
```

7.2.2 Skin Shader

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 GPUGems3 gives you the idea of how to choose diffusion parameters. `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

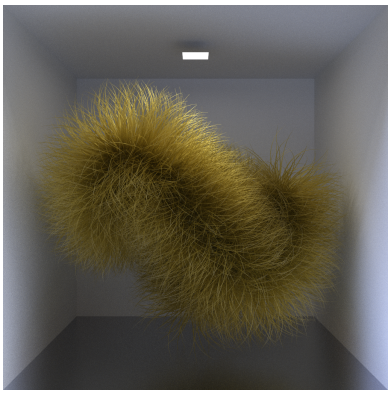


Figure 9: Eugene d'Eon's energy conserving model and automatic hair UVW assignment

Figure 10: 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 for hair.

```
shader
    name "hair"
    surface_shader
        dielectric
        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

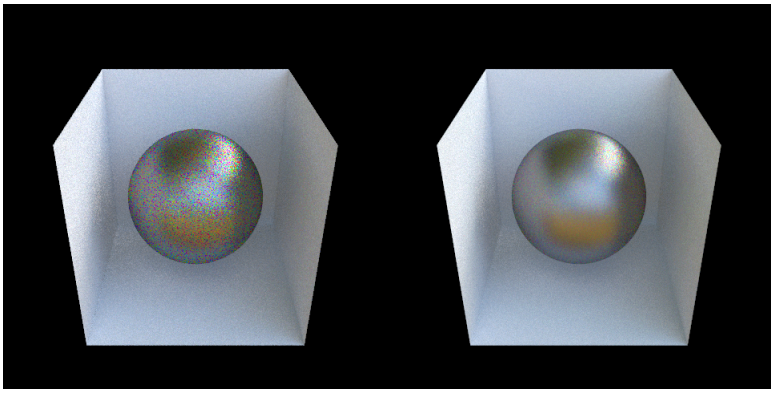


Figure 11: Layering R, G, and B colored glossy components. The left image uses $sample = 4$ and right uses $sample = 31$.

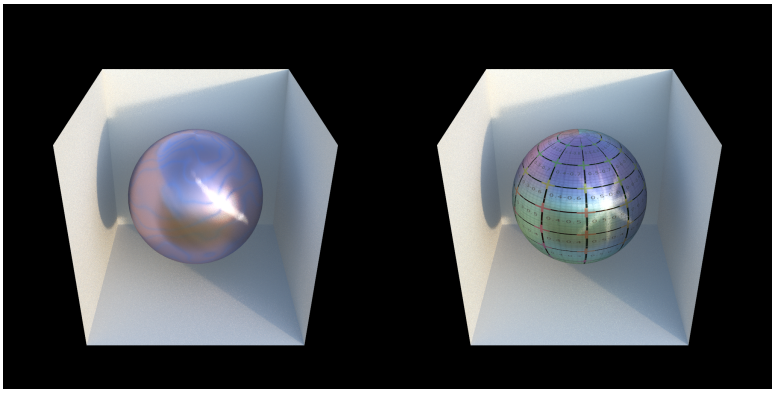


Figure 12: Complicated highlight patterns with *weight* maps.

redqueen supports one shader per primitive but *surface_shader* is layerable. In other words, *shader* can have multiple *surface_shader*s. The maximum number of *surface_shader*s 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_shader*s. 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*.

```
geometry_shader
    smooth_angle 90
    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
```

7.3.1 Options

The currently supported options are: *flip_face* and *smooth_angle*. *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 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.

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* 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.

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 (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

rqInitialize() 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. rqInitializeCameras( );
2. rqInitializeShaders( );
3. rqInitializeAccelerator( );
4. rqInitializeSkyLight( );
5. rqInitializeGeometryLight( );
6. rqInitializeGaffer( );
7. rqInitializeRenderer( );

9.2 Creating Mesh

You can create mesh objects using rqAddObject() and rqAddPart(). *part* is a data structure to support arbitrary vertex data. **redqueen** does not remove duplicated vertices unless you use positive *smooth_angle* so please be aware of mesh sanity.

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

As mentioned earlier, each *part* 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 Sudio, 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.