ORB.S : Open Rendering Benchmark Spec

Ray Tracing Systems

October 8, 2021

Abstract

The standard feature number definition consists of 3 parts. They are material, lights and geometry. Thus to determine a finite set of features renderer support you have to specify 3 numbers in subsequent order. For example: «G0.1/M0.3/L0.4» means «0.1» version for geometry, «0.3» version for materials and «0.4» for lights. Each number in specification definition means all «core» requirenments with less number are fulfilled. For example, materials ver 2.2 means 0.* and 1.* requirenemnts, 2.1 and 2.2 are implemented/fulfilled but 2.3 may be not implemented/fulfilled.

1 List of requirenments

2 Geometry v1.0

General requirenments:

(!)REQ: Renderer should not do any geometry preprocessing (such as normals recalculation, vertex splitting, etc.) and use provided data as it is. This is valid for all specification levels unless otherwise specified.

geo_tri_1.0: primitive_format (core)

The geometry should be represented as triangulated indexed meshes. Indices must be specified with uniform winding order for the whole scene - only clockwise or only counter-clockwise. Triangle vertices **must** have the following attributes specified:

- positions in 3d space;
- normals;
- tangents;
- texture coordinates (can be set to zeroes if are not available).

For each triangle of this mesh some material is assigned — that is, per triangle material is specified.

geo_tri_1.1: single_mesh (core)

At this specification level, the whole scene is defined as a single mesh or several meshes with transformed vertices in world space. All positions of vertices must me specified in meters.

There are five parameters that should be specified for drawing model: (fovY, width, height, zNear, zFar). If they are not specified, renderer may use default values:

- fovY – vertical field of view in degrees, default value is 90.0.
- width – image width in pixels, default value is 1024.
- height – image height in pixels, default value is 1024.

- zNear – near plane in meters, default value is 0.1.
- zFar – far plane in meters, default value is 100.0.

The projection matrix should be obtained from (left,right,bottom,top) in the way OpenGL standart specifies for glFrustum according to further calculations:

$$\begin{aligned} aspect &= \frac{width}{height} \\ ymax &= zNear * tan(fovY * \frac{\pi}{360.0}) \\ xmax &= ymax * aspect \\ left &= -xmax, right = +xmax, bottom = -ymax, top = +ymax \end{aligned}$$

Reference implementation can be found here (perspectiveMatrix)
The camera should be specified in the following way:

- camera position : point in 3D.
- camera lookAt : point in 3D.
- camera upVector : point in 3D, default value is (0,1,0).
- camera distance : scalar, represent distance to focal plane.

The modelView matrix should be calculated from (position,lookAt,upVector) according to gluLookAt specification. Reference implementation can be found here (lookAt).

geo_tri_1.2: inst_matrices (core)

At this specification level, the whole scene is defined as set of instances of meshes. Each mesh instance has its own transform 4x4 matrix. Matrices can contain rotation, translation, scaling (both linear and non-linear). If the mesh was not instanced at all, it should not be drawn.

Test: test_005_instancing.

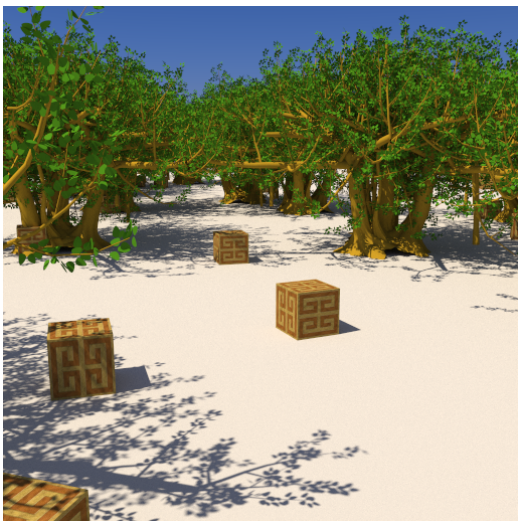


Figure 1: Example of instancing.

geo_tri_1.3: inst_change_material (core)

This requirement defines the ability to instantiate same objects with different materials. However, only single material for the whole mesh (and thus fo the whole instance) is allowed. Though, rasterization based traditional renderers assume to change their shaders once per instance and the ray tracing based solutions should substitute single material id for each instance.

(!)Note: In other words, each mesh has strictly 1 material, and if any of them has 2 different materials, the exporter must divide this mesh into two different meshes.

(!)Note: Here we can see a unobvious thing for content exporter developers. On both previous and subsequent levels of specification for a single mesh it is allowed to have several materials. But current specification level directly forbids this: export/preparation software must split meshes to have only single material per mesh. It is important to emphasize that there is no contradiction here, because our specification is created for renderers and thus exporters must adapt to this. For render systems without full instancing support (geom: **??**, 2) in general there is no difference if object was spitted to several meshes or not, as well remap list (see next spec. level) size may be equal to 1 when we go to subsequent levels.

geo_tri_1.4: inst_remaplist_(MaxMaterials == 4) (core)

This requirement defines the ability to instantiate same objects with different materials. That is, different instances of the same object can have different materials. For this, we define such a thing as «remap list». The remap list is a list of pairs of material indices. In each pair, the first index defines the material to be replaced, and the second index determines the final material. For example, consider further remap lists (listing 1):

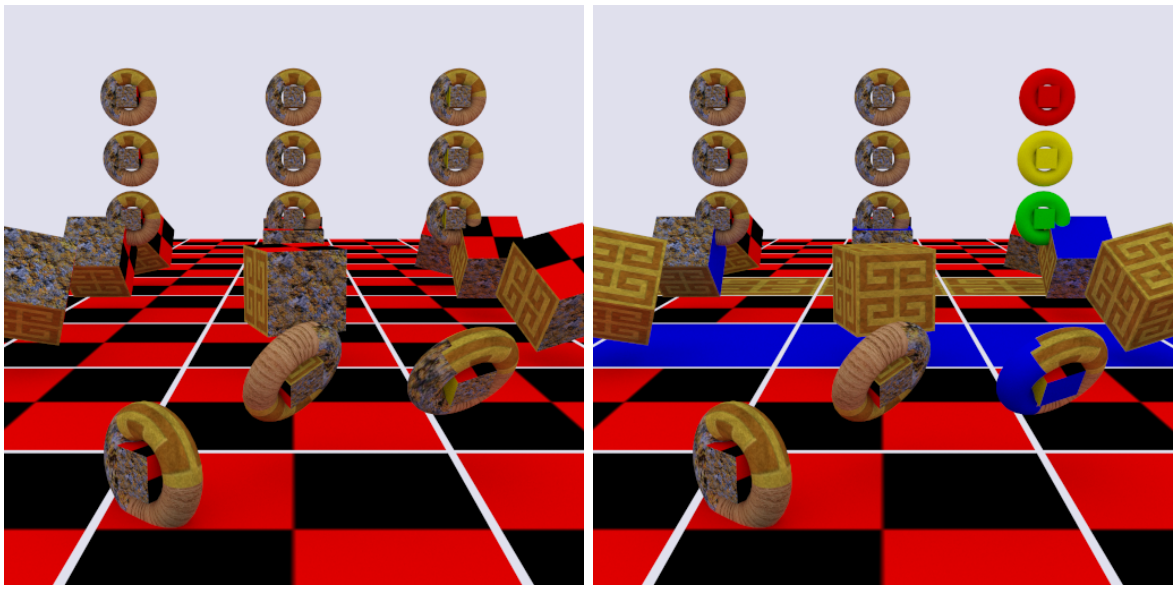
```
<remap_lists>
  <remap_list id="0" size="4" val="0 5 3 1 " />
  <remap_list id="1" size="8" val="1 0 2 0 3 0 4 0 " />
</remap_lists>
```

Listing 1: Remap list example

The remap list with id «0» will replace material 0 to 5 and material 3 to 1. The remap list with id «1» will replace all its input materials (1,2,3,4) to 0.

So, each instance can assign an existing remap list. If this happends, for this instance on the displayed model the replacements specified in remap list should be applied (fig. 4).

Test: test78_material_remap_list1. MSE should be less than 5.0 for 512x512 LDR image.



Original material mapping Appliyng remap list

Figure 2: Remap list example.

(!)Note: For efficiency reasons renderers may restrict remap list size (MaxMaterials == 4). This is essential, for example for objects with opacity (leafes, grass and e.t.c). If remapped material has opacity property, during ray-triangle intersection we have to look up in the opacity texture. However, material id remapping in general case is implemented via binary search which is too expensive if performed for each triangle. In this way renderer may restrict remap list size to 4 or 8 elements (and thus 2 or 4 materials, for example) to be sure that remapping will be cheap enough. Please note that default maximum material amount in remap list is equal to 4 and thus exporter must split single mesh to several meshes for all instances that have greater than 4 materials in their remap list.

2 Materials v1.0.

Image 3 and equation 1 show lighting integral and related angles. These notation will be used further in the document. They assume reverse (i.e. Path Tracing) direction of light by default:

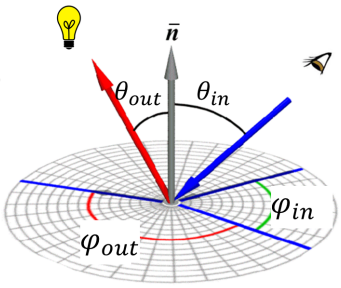


Figure 3: BRDF and angles.

$$I(\phi_{in}, \theta_{in}) = \int_{\phi_{out}} \int_{\theta_{out}} L(\phi_{out}, \theta_{out}) BRDF(\phi_{out}, \theta_{out}, \phi_{in}, \theta_{in}) cos(\theta_{out}) \mathrm{d}\phi_{out} \mathrm{d}\theta_{out} \tag{1}$$

It should be noted that the adopted agreement of the backward propagation of light (from eye to light) does not limit anything and can be applied also for calculation of forward propagation (from light to eye).

(!) Implementation REQ. : All BRDF and BTDF should be implemented without $cos(\theta_{out})$ term. For mirrors cosine shouldn't be applied, thus you should pre-divide $cos(\theta_{out})$ instead (so that the cosines cancel each other). Also note that we do not include color in BRDF/BTDF equations. The color should be multiplied with the BRDF value.

(!) Implementation REQ. : Note that texture color (that comes from PNG, JPG, BMP and othe LDR formats) should be transformed to linear space with gamma transform: $C_{linear} = C_{srgb}^{2.2}$. Unlike textures values that are **specified explicitly** in XML (or other) assummed to be **already in linear space**.

(!) Implementation REQ. : Note that final color of the image should be first clamped and then transformed from HDR to LDR space with gamma correction: $C_{srgb} = C_{linear}^{(1.0/2.2)}$. Clamping should be simple (i.e. losing color). For example color (1.5, 2.0, 10.0) should be clamped to (1,1,1) but not to (0.15, 0.2, 1.0) for example.

(!) Symbols:

- $\omega_i = v$: in, or view vector (note that on image 3 inverse view vector (i.e. $-v$) is drawn with blue arrow);
- $\omega_o = l$: out, or light vector;
- $\omega_h = \frac{l+v}{2}$, half vector;

- n, normal vector.

mat_1.0: lambert (core)

Basic support for Lambertian reflections. The BRDF function **must** be implemented according to equation 2. See fig 4.

Implementation:

$$BRDF_{lambert} = \frac{1}{\pi} \tag{2}$$

Test: 101_diffuse_lambert. MSE should be less than 5.0 for 512x512 LDR image.

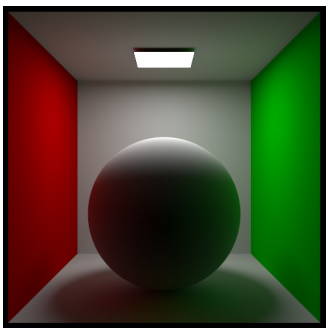
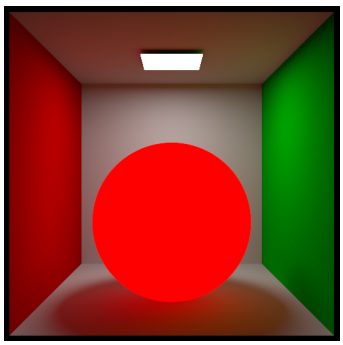


Figure 4: Example of lambertian material. The grey ball have color equal to (0.5, 0.5, 0.5).

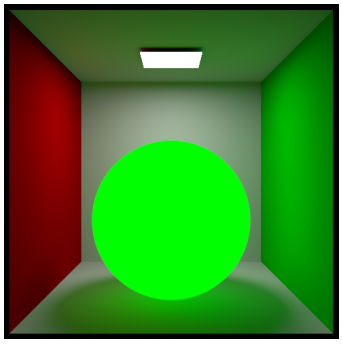
mat_1.1: emission (core)

Basic support for emissive material. On this level of specification emissive materials don't combined with others. Renderer may suppose mutually exclusive: (emissive or 'some other'). Please note that emissive materials have not be considered as lights (i.e. they don't have to be sampled with any explicit sampling strategy, however, of cource, they could), but they must cast GI.

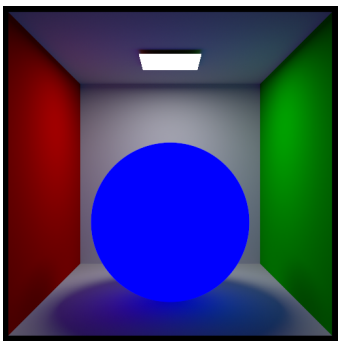
Test: 123_emission. MSE should be less than 5.0 for 512x512 LDR image.



red emission



green emission



blue emission

Figure 5: Phong for different cosine power and color (0.5, 0.5, 0.5).

(!) Implementation REQ. : Emission color value **must** specify **brightness** value. It is easy to ckeck this requirenment. Directly visible emissive surface on rendered image (stored in linear HDR framebuffer) must have exactly same value that was set for material. For example if you set emission color to (0.5, 0.5, 0) and then check directly visible surfaces (when primary rays immediately hit surface), you shoulg get exactly same yellow color (0.5, 0.5, 0) in the rendered image (in linear space, i.e. HDR image).

mat_1.2: perfect_mirror (core)

Pure specular reflection. Note that mixing of mirror and lambertian reflections is not needed at this level of specification. Renderer may suppose mutually exclusive: (phong, mirror or lambert). **Implementation:** It is assumed that you will pre-divide to $cos(\theta_{out})$ inside BRDF and later cancel this cosine in lighting integral (equation 1) instead of considering pure specular reflection as a special case during lighting integral evaluation. Though, you will probably have to consider pure specular reflection as special cases in many other places.

$$BRDF_{mirror} = \frac{1}{cos(\theta_{out})} \tag{3}$$

(!) Implementation Note: You are **allowed** to clamp denominator to avoid dividing by zero:

$$BRDF_{mirror} = \frac{1}{\max(cos(\theta_{out}), 10^{-6})} \tag{4}$$

Test: test_174_mirror_orbspec_mat03.

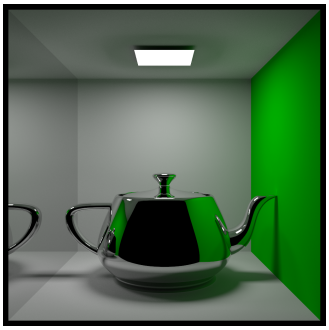


Figure 6: Example of mirror material with color equal to (0.75, 0.75, 0.75).

mat_1.3: ggx (core)

The GGX isotropic BRDF implementation according to [2] and [3]:

$$A = \frac{1}{\omega_i \cdot n + \sqrt{\alpha^2 + (1 - \alpha^2)(\omega_i \cdot n)^2}}$$

$$
\begin{aligned}
B &= \frac{1}{\omega_o \cdot n + \sqrt{\alpha^2 + (1-\alpha^2)(\omega_o \cdot n)^2}} \\
D &= \frac{\alpha^2}{\pi \left((\omega_h \cdot n)^2 . (\alpha^2 - 1) + 1\right)^2} \\
f_r(\omega_o, \omega_i, \alpha) &= A * B * D
\end{aligned}
$$

Where α is roughness and (ω_o, ω_i) described previously here 2.

GGX sampling can be efficiently implemented according to the Eric Heitz paper [3]. Additional implementation with further explanation can be found here [4].

Test: 330_CornellBoxWithSphereCylinderTeapot.

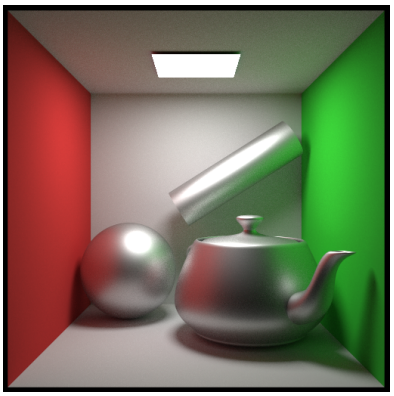


Figure 7: testing GGX: Todo describe parameters, add make images for different roughness cases.

mat_1.4: lambert_texture (core)

Adding LDR textures for Lambertian materials. When reading from LDR texture, gamma transform **should** be applied. The resulting color must be the product of texture color and material color. Texture matrices support is not needed on this level of specification.

Test: test_103_diffuse_texture. MSE should be less than 5.0 for 512x512 LDR image.

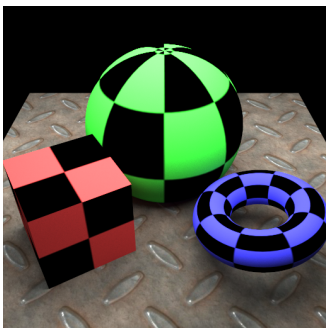


Figure 8: Example of lambertian material with textures.

mat_1.5: emission_texture_ldr (core)

Adding LDR textures to emission color. When reading from LDR texture, gamma transform **should** be applied. The resulting color must be the product of texture color and material color. Texture matrices support is not needed on this level of specification.

Test: test_124_emission_texture. MSE should be less than 5.0 for 512x512 LDR image.

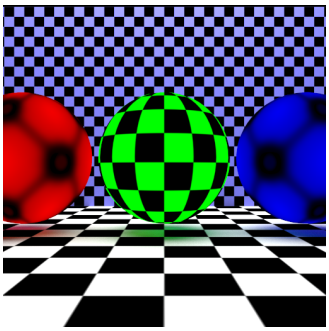


Figure 9: Emissive textures.

mat_1.6: ggx_texture (core)

Adding LDR textures for reflection color for both ggx and mirrors. When reading from LDR texture, gamma transform **should** be applied. The resulting color must be the product of texture color and material color. Texture matrices support is not needed on this level of specification.

Test: test_108_reflect_texture. MSE should be less than 5.0 for 512x512 LDR image.

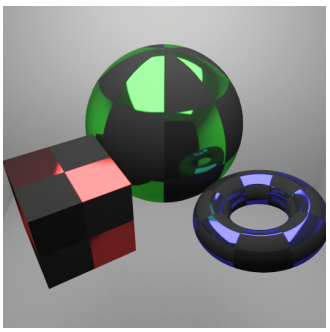


Figure 10: Example of phong material with textures.

(!) Implementation REQ. : Texture fetch (samplers) must support both «wrap» and «clamp» addressing modes separately for u and v texture coordinates. Please see req. 4 for detailed description.

mat_1.7: ggx_texture_glosiness (core)

Adding textures for reflection $glosiness = 1 - roughness$ in ggx model. Colored textures should use $max(color.r, color.g, color.b)$ to get glosiness value from textures.

Note: On this specification level ggx with glosiness equal to 1 **must** become mirror material.

mat_1.8: texture_wrap_and_clamp (core)

Texture fetch (samplers) must support both «wrap» and «clamp» addressing modes separately for u and v texture coordinates. If the mode is not specified, the default mode is wrap. Lack of support for the «clamp» mode in test (req.4) will lead to incorrect visualization of the edges of the cube due to interpolation between red and black colors, which should be absent with the «clamp» mode. Texture sampler parameters must be enabled for further material parameters: diffuse, reflection, emission, glosiness.

mat_1.9: texture_matrices (core)

The sampler object must support the storage and use of texture matrices. Texture matrices are aplied to texture coordinates before actual texture fetch is performed. Initially, the matrices should be 4x4, but since in most cases the texture coordinates are two-dimensional, at this level of the specification, the render can save only the first 2 rows of the matrix.

mat_1.10: mix_ggx_lambert_extrusion_maxcolor (core)

Mixing ggx and lambert. The ggx reflection have a higher priority and **must** extrude Lambertian reflection if summ of reflection coefficients are greater than 1.

Colored textures gain $max(color.r, color.g, color.b)$ for mix coefficient.

mat_1.11: mix_ggx_lambert_extrusion_avgcolor (core)

Mixing ggx and lambert. The ggx reflection have a higher priority and **must** extrude Lambertian reflection if summ of reflection coefficients are greater than 1. The Corona and Mental Ray renderers use this extrusion type by default.

Colored textures gain $avg(color.r, color.g, color.b)$ for mix coefficient.

mat_1.12: mix_ggx_lambert_extrusion_percomponent (extension)

Mixing ggx and lambert separately for each color component. The ggx reflection have a higher priority and **must** extrude lambertian reflection *separately for each color component* if summ of reflection coefficients are greater than 1. The VRay renderer use this extrusion type by default.

mat_1.13: mix_ggx_lambert_extrusion_sumcolor (extension)

Mixing ggx and lambert without priority. If summ of reflection coefficients is greater than 1, both of them **should** be scaled linearly to get sum equal to 1.0 *separately for each color component*.

Note that all mixtures could be (and in general recommended to be) implemented via blend by mask (see blend_mask). However, on this level of specification you don't have to implement blend for BRDFs.

4 Light v1.0

lgt_1.0: area_light (core)

2 Integrators

alg_2.1: paths_basic (core)

Scenes should be tested in which the following types of paths contribute significantly to the image:

1. ESDL, EDSL, ESDSL (R) — mirror wall that is also reflect single reflected caustic;
2. EGDL, EDGL, EGDGL (R) — glossy wall that is also reflect single reflected caustic;
3. ESSL, EDSSL — double perfect reflection.
4. EGGL, EDGGL — double glossy reflection.

alg_2.2: paths_glass (core)

Scenes should be tested in which the following types of paths contribute significantly to the image:

1. EDSL, EDSL (T) — spherical light inside glass ball to check light transport symmetry;
2. ESDSL (T) — single refracted caustic to check light transport symmetry (water scene);
3. ESSL (T) — single refracted caustic to check light transport symmetry (glass ball);

alg_2.3: paths_glossy_glass (core)

Scenes should be tested in which the following types of paths contribute significantly to the image:

1. EDGL, EDGL (T) — spherical light inside glass ball to check light transport symmetry;
2. EGDGL (T) — single refracted caustic to check light transport symmetry (water scene);
3. EGGL (T) — single refracted caustic to check light transport symmetry (glass ball);

References

[1] Eric P. Lafortune and Yves D. Willems. *Using the Modified Phong Reflectance Model for Physically Based Rendering*. 1994.

[2] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. *Microfacet models for refraction through rough surfaces.* In Proceedings of the 18th Eurographics conference on Rendering Techniques (EGSR'07). 2007. Eurographics Association, Goslar, DEU, 195–206.

[3] Eric Heitz. *A Simpler and Exact Sampling Routine for the GGX Distribution of Visible Normals.* [Research Report] Unity Technologies. 2017. hal-01509746

[4] Joe Schutte. *Importance Sampling techniques for GGX with Smith Masking-Shadowing: Part 2.* March 7, 2018. Github pages, accessed 22.05.2020.