
Seurat - Documentation

What is Seurat?

Seurat is a system for image-based scene simplification for VR. It converts complex 3D scenes with millions of triangles, including complex lighting and shading effects, into just tens of thousands of triangles that can be rendered very efficiently on 6DOF devices with little loss in visual quality. It delivers high fidelity graphics on mobile VR devices. (One way to think of it is as a serving the same role as stereo panoramas on 3DoF VR devices, on 6DoF devices.)

The processing pipeline for static environments generates data for a single headbox (e.g. 1 m³ of space). Input data can be generated with any rendering system, e.g. a real-time game engine or an offline ray tracer. We have plugins for Unity, Unreal and Maya. Seurat outputs a mesh with an RGBA texture atlas, which can be rendered in any real-time engine. Dynamic content can be composited on top of the static Seurat environments.

Using Seurat

Using Seurat requires three steps:

1. Generating the RGBD input images from your scene needed for the Seurat processing pipeline.
2. Running the images through the pipeline to generate the output geometry and RGBA texture atlas.
3. Importing the Seurat output into your engine of choice.

This document primarily discusses (1) and (2) – generating the inputs to the pipeline and running it. We provide plugins for Unity and Unreal Engine to simplify steps (1) and (3), and we illustrate capture from raytracers via an Autodesk® Maya® script.

You can access them here:

Seurat Unity Plugin

Seurat Unreal Plugin

Seurat Maya Script

Scene Capture

A scene capture consists of a set of RGBD images and a JSON manifest describing the capture. The capture is organized into *view groups*. A view group is a set of *views*, each consisting of a camera and the associated image data (RGB and depth). The views of a view group must have disjoint camera

frusta. A common setup is to render 32 cube maps (the view groups) from random positions inside the headbox.

The images can be generated with any offline or real-time rendering engine, but they must have the following properties. Each pixel must correspond to the color and depth value of exactly one camera ray. This means that antialiasing needs to be turned off and features such as depth of field and motion blur have to be disabled. The same applies to most screen space effects, e.g. bloom and tone mapping.

Command line parameters

input_path [default=""] [required] Path to the input manifest.json file.

output_path [default=""] [required] Base path to all output artifacts. Filename extensions will be added automatically. E.g. if output_path is foo, the pipeline will produce foo.obj and foo.png.

cache_path [default=""] Directory for all cache artifacts. If empty, no cache will be used. Otherwise, results from the geometry stage will be cached in the specified directory. On subsequent runs, the geometry is loaded from this cache. This is useful for processing multiple texture channels for the same geometry or for iterating on lighting and shading.

single_face [default=""] If not empty, process only the specified face of a cube around the headbox center. Must be one of 'front', 'back', 'left', 'right', 'bottom', 'top'. This is useful for fast previews where a full 360 degree scene is not required.

triangle_count [default=72000] The maximum number of triangles to generate.

overdraw_factor [default=3.0] The target amount of overdraw. Seurat bounds the average overdraw over a full 360 view. E.g. if overdraw_factor is set to 3, and the Seurat output is rendered into a cube map, then every pixel of that cube map will be covered by three quads on average. Seurat can allocate more quads in areas with high-depth complexity (e.g. trees) and less in simpler areas (e.g. solid wall). As a consequence, the maximum overdraw for any particular view direction is not guaranteed to be bounded (we are working on this).

peak_overdraw_factor [default=999.0] The peak overdraw in any view direction. The default value effectively disables this features. Set to a small value (e.g. 5 or 6) to turn it on if overdraw is an issue.

gamma [default=1.0] Gamma-correction exponent.

specular_filter_size [default=0.05] The size of the filter used to 'bake' specular highlights. Smaller values bake sharper reflections. Larger values blur these out, yielding a more diffuse-looking representation. Use "+Infinity" for baking gbuffer values such as normal maps.

premultiply_alpha [default=true] Determines whether output textures use premultiplied alpha.

ray_footprint [default=0.01] The 'footprint' of a sample, along its depth. Larger values help fill & inpaint possible seams in the final geometry.

pixels_per_degree [default=13.0] Resolution of the target display in pixels per degree. This parameter is used to determine texture size. It should be set to the resolution of the target HMD. Note that Seurat may automatically lower the resolution to not exceed the `max_texture_size`.

texture_width [default=4096] The target width of the output texture. If necessary, the resolution in `pixels_per_degree` is reduced automatically to fit the result into an atlas of this size.

texture_height [default=4096] The target height of the output texture. If necessary, the resolution in `pixels_per_degree` is reduced automatically to fit the result into an atlas of this size.

texture_alignment [default=4] Alignment constraint (in pixels) on individual texture tiles in the atlas. Adjust this value as needed to match the alignment requirements of external block-based texture compression methods.

content_adaptive_resolution [default=false] Determines whether to adapt local texture resolution based on texture content.

skybox_radius [default=200.0] Half the side-length of the origin-centered skybox to clamp distant geometry. 0.0 indicates no skybox clamping should be performed.

fast_preview [default=false] Determines whether to prefer speed over quality.

report_progress [default=true] Print progress updates to stdout.

z_buffer [default=false] When integrating Seurat output into an existing rendering pipeline, there are two main options for rendering its geometry: Render Seurat output with alpha blending, without writing to a z-buffer. Render Seurat output with alpha-to-coverage (a.k.a. “alpha to mask”) with z-buffer writes enabled. This flag indicates which rendering mode will be used, and the output will be optimized for rendering with that method.

projective_texture_mapping [default=false] Enables projective texture mapping. Otherwise object space texture mapping is used. Projective texture mapping significantly reduces texture distortion on grazing angle quads.

separate_opaque [default=false] Determines whether separate meshes and texture atlases will be output for opaque and translucent parts of the scene.

alpha_threshold [default=0.95] Defines the threshold for deciding whether a texture is opaque or translucent. A value between 0.0 and 1.0 is expected.

pixel_filter [default=“gaussian”] Pixel filter for texture generation. Must be one of ‘box’ (unit box filter), ‘bspline’ (cubic B-Spline), ‘gaussian’ (truncated Gaussian with radius = 1.5 and sigma = 0.3).

JSON Manifest

Coordinate Spaces

All coordinate spaces are following the OpenGL convention of left-handed coordinate systems and cameras looking down the negative Z-axis. The naming convention for matrices

is `foo_from_bar_matrix` for a matrix that transforms from bar-space to foo-space. E.g. the `world_from_eye_matrix` transforms points or vectors in eye-space into world-space.

Depth Encoding

Seurat currently support three depth encodings: `WINDOW_Z`, `EYE_Z` and `RAY_DEPTH`. Other encodings can be added as needed.

WINDOW_Z Depths are the window-space Z coordinate (Z/W, as in Z buffer from GL) in the range [0.0, 1.0].

EYE_Z Depths are the negated eye-space Z coordinate in the range [0, inf).

RAY_DEPTH Depths are distances along a normalized ray (unit length direction vector) through a pixel center. In other words, this is the distance between the point and the origin in eye-space.

Spec

```
1 Capture := {
2   Point3f headbox_center; // optional
3   ViewGroup view_groups[];
4 }
```

```
1 ViewGroup := {
2   View views[];
3 }
```

```
1 View := {
2   ProjectiveCamera projective_camera;
3   DepthImageFile depth_image_file;
4 }
```

```
1 ProjectiveCamera := {
2   int image_width;
3   int image_height;
4   Matrix4f clip_from_eye_matrix;
5   Matrix4f world_from_eye_matrix;
6   String depth_type := ""WINDOW_Z, ""EYE_Z, ""RAY_DEPTH;
7 }
```

```
1 DepthImageFile := {
2   Image4File color;
3   Image1File depth;
4 }
```

```
1 Image4File := {
```

```
2   String path;
3   String channel_0;
4   String channel_1;
5   String channel_2;
6   String channel_alpha;
7 }
```

```
1 Image1File := {
2   String path;
3   String channel_0;
4 }
```

```
1 Point3f := double[3]
```

```
1 Matrix4f := double[16]
```

Description

The root node is a Capture object.

Matrices are in row-major order.

Channel names in Image4File and Image1File can be arbitrary strings (for OpenEXR input) or the following reserved channel names:

CONSTANT_ZERO Fill channel with 0.0. Supported for all file formats and image types.

CONSTANT_ONE Fill channel with 1.0. Supported for all file formats and image types.

R, G, B, A Use hard-coded color channel from e.g. a PNG file. For OpenEXR input, these names are treated like any other string.

The color and depth images may reference the same file and load from different channels.

The alpha channel is currently only used to mask out pixels if the value is zero. All other values are treated as opaque.

File paths can either be relative to the manifest file, or absolute paths.

Image file formats are automatically detected. Seurat currently supports OpenEXR and PNG.

All views in a view group must have disjoint view frusta. The six faces of a cube map, for example, meet this criterion.

If headbox_center is specified, then all camera world-from-eye matrices are transformed to be relative to this location. This is useful if cameras are specified in world-space.

Example JSON

```
1 {
2   "headbox_center": [0.0, 1.0, 2.0],
3   "view_groups" : [
4     {
5       "views" : [
6         {
7           "projective_camera" : {
8             "image_width" : 1024,
9             "image_height" : 1024,
10            "clip_from_eye_matrix" : [
11              1.0, 0.0, 0.0, 0.0,
12              0.0, 1.0, 0.0, 0.0,
13              0.0, 0.0, 1.0, 0.0,
14              0.0, 0.0, 0.0, 1.0
15            ],
16            "world_from_eye_matrix" : [
17              1.0, 0.0, 0.0, 1.0,
18              0.0, 1.0, 0.0, 2.0,
19              0.0, 0.0, 1.0, 3.0,
20              0.0, 0.0, 0.0, 1.0
21            ],
22            "depth_type" : "EYE_Z"
23          },
24          "depth_image_file" : {
25            "color" : {
26              "path" : "images/0001_front_color.exr",
27              "channel_0" : "R",
28              "channel_1" : "G",
29              "channel_2" : "B",
30              "channel_alpha" : "CONSTANT_ONE"
31            },
32            "depth" : {
33              "path" : "images/0001_front_depth.exr",
34              "channel_0" : "Z"
35            }
36          }
37        }
38      ]
39    }
40  ]
41 }
```

DISCLAIMER: This is not an officially supported Google product.