Arnold Scene Source (ass)

The Structure of .ass Files

Arnold's native scene description file format, known as ".ass" files", are human-readable ASCII text files. An .ass file typically contains cameras, lights, model geometry, and shaders as a list of nodes with their connections and parameters.

The Arnold DCC plug-ins for Maya, Houdini, etc. can export ass files. Arnold's command-line renderer, *kick*, can be used to render these ass files into image files.

An example .ass file

The following is a simple .ass file that contains an options block, a filter node, a driver, a camera, a light, a polymesh, and a shader.

```
options
{
 AA_samples 3
 outputs "RGBA RGBA myfilter mydriver"
 xres 720
 yres 486
}
gaussian_filter
{
 name myfilter
 width 2.0
}
driver_tiff
{
 name mydriver
 filename "image.tif"
 color_space auto
}

persp_camera
{
 name mycamera
 fov 53.638
 matrix
 1 0 -0 0
 -0 0.995 -0.0995 0
 0 0.0995 0.995 0
 0 2 20 1
}

distant_light
{
 name mylight
 matrix
 0.78867512 -0.21132487 -0.57735025 0
 -0.21132487 0.78867512 -0.57735025 0
 0.57735025 0.57735025 0.57735025 0
 1 1 1 1
 color 1 1 1
 intensity 1
 cast_shadows on
}

polymesh
{

 name mysphere
 nsides 6 1 BYTE 3 3 3 3 3 3
 vidxs 18 1 UINT
 3 2 0 2 3 1 4 3 0 3 4 1 2 4 0 4 2 1
 nidxs 18 1 UINT
 0 0 1 0 0 2 3 3 4 3 3 5 6 6 7 6 6 8
 vlist 5 1 VECTOR 0 -4 0 0 4 0 -4 0 0 2 0 3.4641015
 2 0 -3.4641015
```

```
 nlist 9 1 VECTOR
 -0.5 0 0.8660254 -0.44721359 -0.44721359 0.77459669
 -0.44721359 0.44721359 0.77459669 1 0 0
 0.89442718 -0.44721359 0 0.89442718 0.44721359 0
 -0.5 0 -0.8660254 -0.44721359 -0.44721359 -0.77459669
 -0.44721359 0.44721359 -0.77459669
 smoothing on
 matrix
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
 shader myshader
}

standard_surface
{
 name myshader
 base 0.7
 base_color 0 1 0
 specular 0.05
 specular_color 1 1 1
 specular_roughness 0.3
}
```

Just like in Python, one-line comments begin with the '#' character.

Nodes and Parameters

Arnold is built around different types of nodes. For example, there are shader nodes, camera nodes, light nodes, filter nodes, geometry (aka "shape") nodes and output driver nodes. Each node has a number of parameters.

Each type of node is identified by a unique name (options, persp_camera, polymesh, lambert, etc.). You can have as many nodes of each node type as you want. Each node is uniquely identified by a string parameter called "name". To create a node, you write the node type and enclose the definition of the parameter values within brackets. For example:

```
lambert
{
 name my_white_shader
 Kd 0.9
 Kd_color 1 1 1
}
```

Nodes

Here are some of the more important nodes that can be found in an ass file.

- `options` - This node is a container of global rendering options, for example:
  - `xres`, `yres`: Image resolution
  - `AA_samples`: Antialiasing samples
  - `camera`: the active camera. It must point to a valid camera node in the scene
  - `outputs`: This array of strings defines a mapping of AOV channels (or layers) to output drivers. The format of each of the strings is: "<AOV_name> <data_type><filter> <driver>"
- `gaussian_filter` - This is the default type of filter used by `kick`. The filter has a user-specified width which defaults to 2.0 pixels.
- `driver_tiff` - Driver node that can be referenced by the `outputs` parameter in the `options` node. It has a filename parameter for the output file that will store the final rendered image.
- `persp_camera` - Camera node that can be referenced by the camera parameter in the options node. Among other parameters, it has a field-of-view and a camera-to-world matrix that defines the orientation.
- `distant_light` - Distant (or directional) light node with a transformation matrix, color, intensity, etc.
- `polymesh` - This is the most important geometric primitive in the renderer. Some of its parameters are:
  - `vidxs`, `nidxs`, `vlist`, `nlist`: These arrays describe the mesh vertices, normals, and their respective topologies (face indices). If the mesh has UV coordinates, they would be stored in the uvidxs and uvlist parameters.
  - `visibility`, `sidedness`: These are bitmasks that define the visibility and sidedness properties for each ray type (camera, shadow, reflection, etc.).
  - `matrix`: The object-to-world transformation matrix of the mesh.
  - `shader`: Pointer to the shader node that will be executed when shading the object

Parameters

All parameters have a default value, so you don't have to explicitly set all of the available parameters. Default parameter values can be queried with Arnold's command line renderer, `kick`. For example, to find the default value of the parameter `Kd` in the lambert shader, type this:

```
% kick -info lambert.Kd
node: lambert
param: Kd
type: FLOAT
default: 0.7
```

The most common types are BOOL, INT, UINT, ENUM, FLOAT, MATRIX, VECTOR, NODE and STRING.

Arrays

Arrays of a basic type, e.g. VECTOR[ ] or FLOAT[ ], are specified with the following syntax:

```
parameter_name <num_elements> <num_motionblur_keys> <data_type> <elem1> <elem2> <elem3> <elem4> ...
```

For example, the polymesh node has a parameter called `vlist`, the array of points where the polygon vertices are stored:

```
% kick -info polymesh.vlist
node: polymesh
param: vlist
type: VECTOR[]
default: (empty)
```

A polymesh with a single triangle would, therefore, be specified with an array of three VECTORs:

```
polymesh
{
 ...
 vlist 3 1 VECTOR 0 0 0 1 0 0 0 0 1
 ...
}
```

Motion Blur

For parameters that support motion blur, you can define several values for each of the motion blur time samples (or "keys"). The following example shows the list of vertices for one triangle with two motion keys, where the triangle has moved 5 units in the Y direction:

```
vlist 3 2 VECTOR 0 0 0 1 0 0 0 0 1 0 5 0 1 5 0 0 5 1
```

The same vertical motion can be achieved by storing static triangle vertices and providing multiple transformation matrices instead:

```
vlist 3 1 VECTOR 0 0 0 1 0 0 0 0 1
matrix 1 2 MATRIX
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0
0 0 1 0
0 5 0 1
```

- .ass File Examples