pointcloud and instance array

The `pointcloud` location is the simplest geometry type in Katana, and in addition to the standard `geometry.arbitrary.*` and `geometry.point.P` handling KtoA extends it with quite a few attributes to make it useful in a variety of situations.

Added Attributes

- `arnoldStatements.*` point attributes: there are various point rendering controls that can be set via the `ArnoldObjectSettings` node. Of particular note are `point_mode` (which determines the use of some of the other attributes) and `min_pixel_width`.
- `geometry.point.aspect`: one float per point, this controls the aspect ratio of each point to make them appear non-square. This only applies to certain `point_mode` choices.
- `geometry.point.rotation`: one float per point, this controls the rotation away from the camera for each point. This only applies to certain `point_mode` choices.
- `geometry.point.width`: the width (diameter) of each point, one float per point. This is in object-space coordinates.
- `geometry.constantwidth`: if per-point width is not required, this can be set with a single float to have all points use the same width. This is preferred over using `geometry.point.constantwidth`.
- `geometry.point.constantwidth`: if per-point width is not required, this can be set with a single float to have all points use the same width.
- (deprecated) `geometry.constantWidth`: note the capitalized 'W'. If per-point width is not required, this can be set with a single float to have all points use the same width.
- (deprecated) `geometry.point.constantwidth`: note the capitalized 'W'. If per-point width is not required, this can be set with a single float to have all points use the same width.

Point-based instancing

If the attribute `geometry.point.instanceSource` or `geometry.instanceSource` is present, the pointcloud switches to generating point-based instances. The additional attributes that are honored in this mode:

- `geometry.point.instanceSource`: a string per point indicating the `instance source` location to be instanced at each point.
- `geometry.instanceSource`: a string indicating the `instance source` location to be instanced for all points.
- `geometry.point.matrix`: one per point, a 16-float matrix per-point indicating a transformation to be applied to each instance.
- `geometry.point.rotate`: one per point, three floats indicating the Euler rotation about Z-axis, then Y-axis, then X-axis for each instance.
- `geometry.point.quaternion`: one per point, four floats indicating quaternion rotation for each instance in order of (real, i, j, k) components.
- `geometry.point.scale`: one per point, three floats indicating X, Y, and Z scale per instance.
- `geometry.point.instanceSkipIndex` or `geometry.instanceSkipIndex`: an array with int indices referring to points to skip. This is similar to `instance array` locations.

Note that the per-instance transformations are applied in the following order: scale, quaternion rotation, rotation (in XYZ order), translation (`geometry.point.P`), and finally the matrix. All of these (except for `geometry.point.P`) can be omitted.

Instance Arrays

Point-based instancing can also be accomplished using `instance array` locations, which has a different attribute layout than `pointcloud` instancing. The following attributes are honored:

- `geometry.instanceSource`: a string array with `instance source` locations, which is indexed by each entry in `geometry.instanceIndex` if there is more than one instance source.
- `geometry.instanceIndex`: optional, if there is more than one location string in `geometry.instanceSource`, then this needs to have one integer index (into `geometry.instanceSource`) per instance.
- `geometry.instanceSkipIndex`: optional, an integer list of indices to skip and not generate instances.
- `geometry.instanceMatrix`: a transformation matrix (4x4, or 16 double values) with one matrix per instance. Note, at least one transform attribute needs to be present.
- `geometry.instanceTranslate`: translation (3 doubles) per instance. Note, at least one transform attribute needs to be present.
- `geometry.instanceRotateX`: a rotation amount and axis, 4 doubles (angle, axis-x, axis-y, axis-z), presumably with the axis as (1, 0, 0). Note, at least one transform attribute needs to be present.
- `geometry.instanceRotateY`: a rotation amount and axis, 4 doubles (angle, axis-x, axis-y, axis-z), presumably with the axis as (0, 1, 0). Note, at least one transform attribute needs to be present.
- `geometry.instanceRotateZ`: a rotation amount and axis, 4 doubles (angle, axis-x, axis-y, axis-z), presumably with the axis as (0, 0, 1). Note, at least one transform attribute needs to be present.
- `geometry.instanceScale`: a scale amount (3 doubles) per instance. Note, at least one transform attribute needs to be present.

Note that the per-instance transformations are applied in the following order: scaling, X rotation, Y rotation, Z rotation, translation, and finally the matrix. At least one per-instance transformation attribute needs to be included.

Common Instancing Attributes

Finally, arbitrary user data (`geometry.arbitrary.*`) attribute groups are also applied. If the scope is `point` (Arnold `varying`) then it will apply each varying value as `primitive` scope (Arnold `constant`) user data on each instance. In this way, arbitrary data can be doled out to each individual instance to customize them. Abitrary data can be explicitly restricted to one (or more) of the instance source children by specifying the `geometry.arbitrary.paramname.match` string array, where each string is a subpath to the particular instance source child you want the arbitrary data to apply to. E.g. if the instance source has children `prim, primgroup/prim1 and primgroup/prim2`, and the `p ointcloud` location has `geometry.arbitrary.myAttr.match` with a string `primgroup/prim1` in it, that arbitrary data will only be applied to the instance created from `primgroup/prim1`.